Tektronix®

8560 MULTI-USER SOFTWARE DEVELOPMENT UNIT TEXT PROCESSING PACKAGE

USERS MANUAL

Tektronix®

COMMITTED TO EXCELLENCE

This manual supports the following TEKTRONIX products:

8560 Option	Product
4A	8560U01

These modules are compatible with:

TNIX Version 1 (8560)

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL.

8560 MULTI-USER SOFTWARE DEVELOPMENT UNIT TEXT PROCESSING PACKAGE USERS MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number ____________________

070-4272-00
Product Group 61

First Printing FEB 1982

ABOUT WARRANTY AND SUPPORT FOR THIS PRODUCT

This product is provided by Tektronix as Category C software.

NOTE

Licensed Software for which the software support is specified as Category C is furnished without warranty of any kind, and without any representation regarding quality, performance, or suitability.

TEKTRONIX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Tektronix' liability for damages, if any, whether based upon contract, negligence, strict liability in tort, warranty, or any other basis, shall not exceed the fee paid by the Customer for the Licensed Software.

Category C software is provided on an "as is" basis. Any software services, if available, will be provided at the then current charges.

CONTENTS

Section 1
INTRODUCTION

TABLES

Section 1
INTRODUCTION

ABOUT THIS PRODUCT

The 8560 MUSDU Text Processing Package is a set of software tools used to prepare documentation necessary to support a development project. The Text Processing Package includes a document formatter for typewriter-like terminals or printers, and a typesetter driver, for phototypesetters. Also included are utilities for typesetting mathematical equations, building and searching reference indices, and building tabular data.

ABOUT THIS MANUAL

This users manual provides tutorial and reference material for use with the 8560 Text Processing Package. The following sections are included:

Installation. Tells you how to install the Text Processing Package software.

Technical Notes. Describes any limitations or special instructions for the programs, and any changes made to the programs by Tektronix.

The NROFF and TROFF Text Processors. Explains how to use the NROFF/TROFF text formatter/typesetter-driver.

Using the -MS Macros with TROFF and NROFF. Explains how to use the MS macro package to tailor NROFF/TROFF to your particular needs.

A TROFF Tutorial. Explains how to use the TROFF typesetter-driver.

EQN—A System for Typesetting Mathematics. Explains how to typeset mathematical equations and formulas.

TBL—A Program to Format Tables. Explains how to format and typeset tabular information easily.

REFER—Some Applications of Inverted Indexes. Explains how to find and insert literature references in your documents.

SOURCE OF DOCUMENTS

The tutorial and reference documents contained in Sections 4 through 9 of this manual are reprinted by permission of Bell Laboratories. The addenda in Sections 4 and 5 are supplied by Tektronix.

LIST OF COMMANDS

Table 1-1 contains a list of the commands included in this package, a brief description of the command's function, and a reference to more detailed information about the command.

Table 1-1
8560 Text Processing Package Commands

Command	Description	Reference
checkeq	Reports errors in eqn constructs.	8560 MUSDU Reference Manual Section 6.
col	Filters reverse linefeeds.	8560 MUSDU Reference Manual Section 6.
deroff	Removes nroff, troff, eqn, and tbl constructs from a file.	8560 MUSDU Reference Manual Section 6.
eqn	Formats mathematical equations for typesetting.	See section 7 of this manual; also see 8560 MUSDU Reference Manual Sections 6 and 7.
look	Searches for lines in a sorted list.	8560 MUSDU Reference Manual Section 6.
lookbib	Finds and inserts literature references in documents.	8560 MUSDU Reference Manual Section 6.
neqn	Formats mathematical equations for output to a terminal.	See section 7 of this manual; also see 8560 MUSDU Reference Manual Section 6.
nroff	Formats and typesets text.	See section 4 and 5 of this manual; also see 8560 MUSDU Reference Manual Section 6.
ptx	Produces a permuted index.	8560 MUSDU Reference Manual Section 6.
pubindex	Produces an inverted bibliographic index	8560 MUSDU Reference Manual Section 6.
refer	Finds and inserts literature references in documents.	See section 9 of this manual; also see 8560 MUSDU Reference Manual Section 6.
spell	Finds spelling errors.	8560 MUSDU Reference Manual Section 6.
tbl	Formats tables for nroff and troff.	See section 8 of this manual; also see 8560 MUSDU Reference Manual Section 6.
tc	Simulates a phototypesetter for TEKTRONIX 4014 or 4015 terminals.	8560 MUSDU Reference Manual Section 6.
troff	Formats text for output to a phototypesetter.	See section 6 of this manual; also see 8560 MUSDU Reference Manual Section 6.

Section 2
INSTALLATION

TABLES

Section 2
INSTALLATION

INTRODUCTION

This section explains the procedure for installing the 8560 Text Processing Package on your 8560 system. The following information is included here: an explanation of the format of the installation disk, installation procedures, and a list of the files needed by each of the Text Processing Package commands.

INSTALLATION PROCEDURES

The Text Processing Package software resides on a flexible disk. The information on the disk consists of executable binary files in **fbr** format. You can load these programs onto your 8560 system disk as a group or you can install individual programs. To load the whole package, use the 8560 command **install.** The **install** command takes all of the information from a **fbr** format disk and loads it to the system disk. If you want to install a single program from the disk, the Command install -f -x program loads the specified program from the **fbr** disk to the system disk.

For each of the Text Processing Package programs to execute properly, certain files must be on the system disk. Refer to the "Dependency Files" discussion later in this section for a complete list of these files. In order for these programs to be installed as system commands, they must be loaded while you are logged in as root.

Installing the Text Processing Package

The general procedure for installing the Text Processing Package is:

1. Log in to the 8560 as root. You must have superuser status to perform the installation.
2. Load the software installation disk into the disk drive.
3. Enter the following command to install the software.

```
# install
```

Installing an Individual Program

The general procedure for installing a particular program from the installation disk is:

1. Log in to the 8560 as root. You must have superuser status to perform the installation.
2. Load the software installation disk into the disk drive.
3. Enter the following command to install the particular program:

   # install -f -x program

For example, to install **tbl** you would enter:

# install -f -x tbl

DEPENDENCY FILES

Table 2-1 lists each command and the files that it needs for execution. The TNIX operating system must also be installed. These files may be installed separately to rebuild a command.

Table 2-1
Files Required for Text Processing Package Commands

Command	Files Required	
checkeq	/bin/checkeq	
col	/bin/col	
deroff	/bin/deroff	
eqn	/bin/eqn	
look	/bin/look	/usr/dict/words
lookbib	/bin/lookbib /usr/lib/refer/hunt	/usr/dict/papers /usr/lib/refer/mkey
neqn	/bin/neqn	
nroff	/bin/nroff /usr/lib/term/tab300-12 /usr/lib/term/tab300s-12 /usr/lib/term/tab4025 /usr/lib/term/tab450-12 /usr/lib/term/tab832 /usr/lib/term/tabascii /usr/lib/term/tablp-8 /usr/lib/term/tablp-t-8 /usr/lib/term/tabq-10 /usr/lib/term/tabq-lg /usr/lib/term/tabq-lg-8 /usr/lib/term/tabvt100 /usr/lib/tmac/tmac.s	/usr/lib/term/tab300 /usr/lib/term/tab300s /usr/lib/term/tab37 /usr/lib/term/tab450 /usr/lib/term/tab450-12-8 /usr/lib/term/taba1 /usr/lib/term/tablp /usr/lib/term/tablp-t /usr/lib/term/tabq /usr/lib/term/tabq-8 /usr/lib/term/tabq-lg-10 /usr/lib/term/tabup /usr/lib/tmac/tmac.an /usr/lib/tmac/tmac.s2c

Table 2-1 (cont)

Command	Files Required	
	/usr/lib/tmac/tmac.sconfid /usr/lib/tmac/tmac.seqn /usr/lib/tmac/tmac.sindex /usr/lib/tmac/tmac.skeep /usr/lib/tmac/tmac.srelpap /usr/lib/tmac/tmac.stechrep /usr/lib/tmac/tmac.stoc	/usr/lib/tmac/tmac.sdisp /usr/lib/tmac/tmac.sfoots /usr/lib/tmac/tmac.sioc /usr/lib/tmac/tmac.srefs /usr/lib/tmac/tmac.stable /usr/lib/tmac/tmac.stitlep
ptx	/bin/ptx /usr/lib/eign	
pubindex	/bin/pubindex /usr/lib/refer/mkey	/usr/lib/refer/inv
refer	/bin/refer /usr/lib/eign /usr/lib/refer/inv	/usr/dict/papers /usr/lib/refer/hunt /usr/lib/refer/mkey
spell	/bin/deroff /bin/spell /usr/dict/hlistb /usr/dict/makefile /usr/dict/spellout /usr/lib/spell	/bin/sed /usr/dict/hlista /usr/dict/hstop /usr/dict/spellin /usr/dict/words
tbl	/bin/tbl	
tc	/bin/tc	
troff	/bin/troff /usr/lib/font/ftBC /usr/lib/font/ftCE /usr/lib/font/ftCK /usr/lib/font/ftCW /usr/lib/font/ftGI /usr/lib/font/ftGR /usr/lib/font/ftL /usr/lib/font/ftPA /usr/lib/font/ftPI /usr/lib/font/ftS /usr/lib/font/ftSI /usr/lib/font/ftUD /usr/lib/tmac/tmac.an /usr/lib/tmac/tmac.s2c /usr/lib/tmac/tmac.sdisp /usr/lib/tmac/tmac.sfoots /usr/lib/tmac/tmac.sioc /usr/lib/tmac/tmac.srefs /usr/lib/tmac/tmac.stable /usr/lib/tmac/tmac.stitlep	/usr/lib/font/ftB /usr/lib/font/ftC /usr/lib/font/ftCI /usr/lib/font/ftCS /usr/lib/font/ftG /usr/lib/font/ftGM /usr/lib/font/ftI /usr/lib/font/ftLI /usr/lib/font/ftPB /usr/lib/font/ftR /usr/lib/font/ftSB /usr/lib/font/ftSM /usr/lib/font/ftXM /usr/lib/tmac/tmac.s /usr/lib/tmac/tmac.sconfid /usr/lib/tmac/tmac.seqn /usr/lib/tmac/tmac.sindex /usr/lib/tmac/tmac.skeep /usr/lib/tmac/tmac.srelpap /usr/lib/tmac/tmac.stechrep /usr/lib/tmac/tmac.stoc

Section 3
TECHNICAL NOTES

This section is reserved for technical information about the 8560 MUSDU Text Processing Package. At the time of this writing, no technical notes are included. Technical notes will be incorporated into later versions of this manual, as needed.

Section 4

THE NROFF AND TROFF TEXT PROCESSORS

INTRODUCTION

The *nroff* and *troff* text processors were developed at Bell Laboratories and are licensed by Western Electric for use on the 8560. The remainder of this section contains a reprint of an article describing *nroff* and *troff* and an addendum detailing the differences between the Bell Laboratories version and the version supplied by Tektronix. The Technical Notes section of this manual describes the limitations of these programs and any changes made to these programs by Tektronix.

NROFF/TROFF User's Manual

Joseph F. Ossanna

Bell Laboratories
Murray Hill, New Jersey 07974

Introduction

NROFF and TROFF are text processors under the PDP-11 UNIX Time-Sharing System[1] that format text for typewriter-like terminals and for a Graphic Systems phototypesetter, respectively. They accept lines of text interspersed with lines of format control information and format the text into a printable, paginated document having a user-designed style. NROFF and TROFF offer unusual freedom in document styling, including: arbitrary style headers and footers; arbitrary style footnotes; multiple automatic sequence numbering for paragraphs, sections, etc; multiple column output; dynamic font and point-size control; arbitrary horizontal and vertical local motions at any point; and a family of automatic overstriking, bracket construction, and line drawing functions.

NROFF and TROFF are highly compatible with each other and it is almost always possible to prepare input acceptable to both. Conditional input is provided that enables the user to embed input expressly destined for either program. NROFF can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

Usage

The general form of invoking NROFF (or TROFF) at UNIX command level is

nroff *options files* (or **troff** *options files*)

where *options* represents any of a number of option arguments and *files* represents the list of files containing the document to be formatted. An argument consisting of a single minus (−) is taken to be a file name corresponding to the standard input. If no fiie names are given input is taken from the standard input. The options, which may appear in any order so long as they appear before the files, are:

Option	*Effect*
−o*list*	Print only pages whose page numbers appear in *list*, which consists of comma-separated numbers and number ranges. A number range has the form *N−M* and means pages *N* through *M;* a initial −*N* means from the beginning to page *N;* and a final *N*− means from *N* to the end.
−n*N*	Number first generated page *N*.
−s*N*	Stop every *N* pages. NROFF will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. TROFF will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and will resume after the phototypesetter START button is pressed.
−m*name*	Prepends the macro file **/usr/lib/tmac.***name* to the input *files*.
−r*aN*	Register *a* (one-character) is set to *N*.
−i	Read standard input after the input files are exhausted.
−q	Invoke the simultaneous input-output mode of the **rd** request.

NROFF Only

−T*name*	Specifies the name of the output terminal type. Currently defined names are **37** for the (default) Model 37 Teletype®, **tn300** for the GE TermiNet 300 (or any terminal without half-line capabilities), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).
−e	Produce equally-spaced words in adjusted lines, using full terminal resolution.

TROFF Only

−t	Direct output to the standard output instead of the phototypesetter.
−f	Refrain from feeding out paper and stopping phototypesetter at the end of the run.
−w	Wait until phototypesetter is available, if currently busy.
−b	TROFF will report whether the phototypesetter is busy or available. No text processing is done.
−a	Send a printable (ASCII) approximation of the results to the standard output.
−p*N*	Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elasped time.
−g	Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output.

Each option is invoked as a separate argument; for example,

nroff **−o***4,8−10* **−T***300S* **−m***abc* *file1* *file2*

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named *file1* and *file2*, specifies the output terminal as a DASI-300S, and invokes the macro package *abc*.

Various pre- and post-processors are available for use with NROFF and TROFF. These include the equation preprocessors NEQN and EQN[2] (for NROFF and TROFF respectively), and the table-construction preprocessor TBL[3]. A reverse-line postprocessor COL[4] is available for multiple-column NROFF output on terminals without reverse-line ability; COL expects the Model 37 Teletype escape sequences that NROFF produces by default. TK[4] is a 37 Teletype simulator postprocessor for printing NROFF output on a Tektronix 4014. TCAT[4] is phototypesetter-simulator postprocessor for TROFF that produces an approximation of phototypesetter output on a Tektronix 4014. For example, in

tbl *files* | **eqn** | **troff** **−t** *options* | **tcat**

the first | indicates the piping of TBL's output to EQN's input; the second the piping of EQN's output to TROFF's input; and the third indicates the piping of TROFF's output to TCAT. GCAT[4] can be used to send TROFF (**−g**) output to the Murray Hill Computation Center.

The remainder of this manual consists of: a Summary and Index; a Reference Manual keyed to the index; and a set of Tutorial Examples. Another tutorial is [5].

Joseph F. Ossanna

References

[1] K. Thompson, D. M. Ritchie, *UNIX Programmer's Manual,* Sixth Edition (May 1975).

[2] B. W. Kernighan, L. L. Cherry, *Typesetting Mathematics — User's Guide (Second Edition),* Bell Laboratories internal memorandum.

[3] M. E. Lesk, *Tbl — A Program to Format Tables,* Bell Laboratories internal memorandum.

[4] Internal on-line documentation, on UNIX.

[5] B. W. Kernighan, *A TROFF Tutorial,* Bell Laboratories internal memorandum.

SUMMARY AND INDEX

Request Form	*Initial Value**	*If No Argument*	*Notes#*	*Explanation*
1. General Explanation				
2. Font and Character Size Control				
.ps ±*N*	10 point	previous	E	Point size; also \s±*N*.†
.ss *N*	12/36 em	ignored	E	Space-character size set to *N*/36 em.†
.cs *FNM*	off	-	P	Constant character space (width) mode (font *F*).†
.bd *F N*	off	-	P	Embolden font *F* by *N*−1 units.†
.bd S *F N*	off	-	P	Embolden Special Font when current font is *F*.†
.ft *F*	Roman	previous	E	Change to font *F* = *x*, *xx*, or 1-4. Also \f*x*, \f(*xx*, \f*N*.
.fp *N F*	R,I,B,S	ignored	-	Font named *F* mounted on physical position 1≤*N*≤4.
3. Page Control				
.pl ±*N*	11 in	11 in	v	Page length.
.bp ±*N*	*N*=1	-	B‡,v	Eject current page; next page number *N*.
.pn ±*N*	*N*=1	ignored	-	Next page number *N*.
.po ±*N*	0; 26/27 in	previous	v	Page offset.
.ne *N*	-	*N*=1 *V*	D,v	Need *N* vertical space (*V* = vertical spacing).
.mk *R*	none	internal	D	Mark current vertical place in register *R*.
.rt ±*N*	none	internal	D,v	Return *(upward only)* to marked vertical place.
4. Text Filling, Adjusting, and Centering				
.br	-	-	B	Break.
.fi	fill	-	B,E	Fill output lines.
.nf	fill	-	B,E	No filling or adjusting of output lines.
.ad *c*	adj,both	adjust	E	Adjust output lines with mode *c*.
.na	adjust	-	E	No output line adjusting.
.ce *N*	off	*N*=1	B,E	Center following *N* input text lines.
5. Vertical Spacing				
.vs *N*	1/6in;12pts	previous	E,p	Vertical base line spacing (*V*).
.ls *N*	*N*=1	previous	E	Output *N*−1 *V*s after each text output line.
.sp *N*	-	*N*=1 *V*	B,v	Space vertical distance *N in either direction.*
.sv *N*	-	*N*=1 *V*	v	Save vertical distance *N*.
.os	-	-	-	Output saved vertical distance.
.ns	space	-	D	Turn no-space mode on.
.rs	-	-	D	Restore spacing; turn no-space mode off.
6. Line Length and Indenting				
.ll ±*N*	6.5 in	previous	E,m	Line length.
.in ±*N*	*N*=0	previous	B,E,m	Indent.
.ti ±*N*	-	ignored	B,E,m	Temporary indent.
7. Macros, Strings, Diversion, and Position Traps				
.de *xx yy*	-	*.yy*=..	-	Define or redefine macro *xx*; end at call of *yy*.
.am *xx yy*	-	*.yy*=..	-	Append to a macro.
.ds *xx string*	-	ignored	-	Define a string *xx* containing *string*.
.as *xx string*	-	ignored	-	Append *string* to string *xx*.

*Values separated by ";" are for NROFF and TROFF respectively.

#Notes are explained at the end of this Summary and Index

†No effect in NROFF.

‡The use of " ' " as control character (instead of ".") suppresses the break function.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.rm *xx*	-	ignored	-	Remove request, macro, or string.
.rn *xx yy*	-	ignored	-	Rename request, macro, or string *xx* to *yy*.
.di *xx*	-	end	D	Divert output to macro *xx*.
.da *xx*	-	end	D	Divert and append to *xx*.
.wh *N xx*	-	-	v	Set location trap; negative is w.r.t. page bottom.
.ch *xx N*	-	-	v	Change trap location.
.dt *N xx*	-	off	D,v	Set a diversion trap.
.it *N xx*	-	off	E	Set an input-line count trap.
.em *xx*	none	none	-	End macro is *xx*.

8. Number Registers

.nr *R* ±*N M*		-	u	Define and set number register *R*; auto-increment by *M*.
.af *R c*	arabic	-	-	Assign format to register *R* (*c*=**1**, **i**, **I**, **a**, **A**).
.rr *R*	-	-	-	Remove register *R*.

9. Tabs, Leaders, and Fields

.ta *Nt* ...	0.8; 0.5in	none	E,m	Tab settings; *left* type, unless *t*=**R**(right), **C**(centered).
.tc *c*	none	none	E	Tab repetition character.
.lc *c*	.	none	E	Leader repetition character.
.fc *a b*	off	off	-	Set field delimiter *a* and pad character *b*.

10. Input and Output Conventions and Character Translations

.ec *c*	\	\	-	Set escape character.
.eo	on	-	-	Turn off escape character mechanism.
.lg *N*	-; on	on	-	Ligature mode on if *N*>0.
.ul *N*	off	*N*=1	E	Underline (italicize in TROFF) *N* input lines.
.cu *N*	off	*N*=1	E	Continuous underline in NROFF; like **ul** in TROFF.
.uf *F*	Italic	Italic	-	Underline font set to *F* (to be switched to by **ul**).
.cc *c*	.	.	E	Set control character to *c*.
.c2 *c*	'	'	E	Set nobreak control character to *c*.
.tr *abcd*....	none	-	O	Translate *a* to *b*, etc. on output.

11. Local Horizontal and Vertical Motions, and the Width Function

12. Overstrike, Bracket, Line-drawing, and Zero-width Functions

13. Hyphenation.

.nh	hyphenate	-	E	No hyphenation.
.hy *N*	hyphenate	hyphenate	E	Hyphenate; *N* = mode.
.hc *c*	**\%**	**\%**	E	Hyphenation indicator character *c*.
.hw *word1* ...		ignored	-	Exception words.

14. Three Part Titles.

.tl *'left'center'right'*		-	-	Three part title.
.pc *c*	**%**	off	-	Page number character.
.lt ±*N*	6.5 in	previous	E,m	Length of title.

15. Output Line Numbering.

.nm ±*N M S I*		off	E	Number mode on or off, set parameters.
.nn *N*	-	*N*=1	E	Do not number next *N* lines.

16. Conditional Acceptance of Input

.if *c anything*		-	-	If condition *c* true, accept *anything* as input, for multi-line use \\{*anything*\\}.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.if !*c anything*		-	-	If condition *c* false, accept *anything.*
.if *N anything*		-	**u**	If expression $N > 0$, accept *anything.*
.if !*N anything*		-	**u**	If expression $N \leqslant 0$, accept *anything.*
.if *'string1'string2' anything*			-	If *string1* identical to *string2*, accept *anything.*
.if !*'string1'string2' anything*			-	If *string1* not identical to *string2*, accept *anything.*
.ie *c anything*		-	**u**	If portion of if-else; all above forms (like **if**).
.el *anything*		-	-	Else portion of if-else.
17. Environment Switching.				
.ev *N*	*N*=0	previous	-	Environment switched (*push down*).
18. Insertions from the Standard Input				
.rd *prompt*	-	*prompt*=BEL	-	Read insertion.
.ex	-	-	-	Exit from NROFF/TROFF.
19. Input/Output File Switching				
.so *filename*		-	-	Switch source file *(push down).*
.nx *filename*		end-of-file	-	Next file.
.pi *program*		-	-	Pipe output to *program* (NROFF only).
20. Miscellaneous				
.mc *c N*	-	off	E,m	Set margin character *c* and separation *N.*
.tm *string*	-	newline	-	Print *string* on terminal (UNIX standard message output).
.ig *yy*	-	*.yy*=..	-	Ignore till call of *yy.*
.pm *t*	-	all	-	Print macro names and sizes; if *t* present, print only total of sizes.
.fl	-	-	B	Flush output buffer.
21. Output and Error Messages				

Notes-

B	Request normally causes a break.
D	Mode or relevant parameters associated with current diversion level.
E	Relevant parameters are a part of the current environment.
O	Must stay in effect until logical output.
P	Mode must be still or again in effect at the time of physical output.
v,p,m,u	Default scale indicator; if not specified, scale indicators are *ignored.*

Alphabetical Request and Section Number Cross Reference

ad 4	cc 10	ds 7	fc 9	ie 16	ll 6	nh 13	pi 19	rn 7	ta 9	vs 5
af 8	ce 4	dt 7	fi 4	if 16	ls 5	nm 15	pl 3	rr 8	tc 9	wh 7
am 7	ch 7	ec 10	fl 20	ig 20	lt 14	nn 15	pm 20	rs 5	ti 6	
as 7	cs 2	el 16	fp 2	in 6	mc 20	nr 8	pn 3	rt 3	tl 14	
bd 2	cu 10	em 7	ft 2	it 7	mk 3	ns 5	po 3	so 19	tm 20	
bp 3	da 7	eo 10	hc 13	lc 9	na 4	nx 19	ps 2	sp 5	tr 10	
br 4	de 7	ev 17	hw 13	lg 10	ne 3	os 5	rd 18	ss 2	uf 10	
c2 10	di 7	ex 18	hy 13	li 10	nf 4	pc 14	rm 7	sv 5	ul 10	

Escape Sequences for Characters, Indicators, and Functions

Section Reference	*Escape Sequence*	*Meaning*
10.1	\\	\ (to prevent or delay the interpretation of \)
10.1	**\e**	Printable version of the *current* escape character.
2.1	\'	' (acute accent); equivalent to \(**aa**
2.1	\`	` (grave accent); equivalent to \(**ga**
2.1	\-	− Minus sign in the *current* font
7	\.	Period (dot) (see **de**)
11.1	\(space)	Unpaddable space-size space character
11.1	**\0**	Digit width space
11.1	\\|	1/6 em narrow space character (zero width in NROFF)
11.1	\^	1/12 em half-narrow space character (zero width in NROFF)
4.1	**\&**	Non-printing, zero width character
10.6	\!	Transparent line indicator
10.7	\"	Beginning of comment
7.3	**\$***N*	Interpolate argument $1 \leqslant N \leqslant 9$
13	**\%**	Default optional hyphenation character
2.1	\(*xx*	Character named *xx*
7.1	*****x*, *****(xx*	Interpolate string *x* or *xx*
9.1	**\a**	Non-interpreted leader character
12.3	**\b**'*abc...*'	Bracket building function
4.2	**\c**	Interrupt text processing
11.1	**\d**	Forward (down) 1/2 em vertical motion (1/2 line in NROFF)
2.2	**\f***x*,**\f(***xx*,**\f***N*	Change to font named *x* or *xx*, or position *N*
11.1	**\h**'*N*'	Local horizontal motion; move right *N (negative left)*
11.3	**\k***x*	Mark horizontal *input* place in register *x*
12.4	**\l**'*Nc*'	Horizontal line drawing function (optionally with *c*)
12.4	**\L**'*Nc*'	Vertical line drawing function (optionally with *c*)
8	**\n***x*,**\n(***xx*	Interpolate number register *x* or *xx*
12.1	**\o**'*abc...*'	Overstrike characters *a, b, c, ...*
4.1	**\p**	Break and spread output line
11.1	**\r**	Reverse 1 em vertical motion (reverse line in NROFF)
2.3	**\s***N*,**\s**±*N*	Point-size change function
9.1	**\t**	Non-interpreted horizontal tab
11.1	**\u**	Reverse (up) 1/2 em vertical motion (1/2 line in NROFF)
11.1	**\v**'*N*'	Local vertical motion; move down *N (negative up)*
11.2	**\w**'*string*'	Interpolate width of *string*
5.2	**\x**'*N*'	Extra line-space function *(negative before, positive after)*
12.2	**\z***c*	Print *c* with zero width (without spacing)
16	\{	Begin conditional input
16	\}	End conditional input
10.7	\(newline)	Concealed (ignored) newline
-	\X	*X*, any character *not* listed above

The escape sequences \\, \., \", **\$**, **\***, **\a**, **\n**, **\t**, and \(newline) are interpreted in *copy mode* (§7.2).

Predefined General Number Registers

Section Reference	*Register Name*	*Description*
3	**%**	Current page number.
11.2	**ct**	Character type (set by *width* function).
7.4	**dl**	Width (maximum) of last completed diversion.
7.4	**dn**	Height (vertical size) of last completed diversion.
-	**dw**	Current day of the week (1-7).
-	**dy**	Current day of the month (1-31).
11.3	**hp**	Current horizontal place on *input* line.
15	**ln**	Output line number.
-	**mo**	Current month (1-12).
4.1	**nl**	Vertical position of last printed text base-line.
11.2	**sb**	Depth of string below base line (generated by *width* function).
11.2	**st**	Height of string above base line (generated by *width* function).
-	**yr**	Last two digits of current year.

Predefined Read-Only Number Registers

Section Reference	*Register Name*	*Description*
7.3	**.$**	Number of arguments available at the current macro level.
-	.A	Set to 1 in TROFF, if **−a** option used; always 1 in NROFF.
11.1	.H	Available horizontal resolution in basic units.
-	.T	Set to 1 in NROFF, if **−T** option used; always 0 in TROFF.
11.1	.V	Available vertical resolution in basic units.
5.2	**.a**	Post-line extra line-space most recently utilized using **\x**′*N*′.
-	**.c**	Number of *lines* read from current input file.
7.4	**.d**	Current vertical place in current diversion; equal to **nl**, if no diversion.
2.2	**.f**	Current font as physical quadrant (1-4).
4	**.h**	Text base-line high-water mark on current page or diversion.
6	**.i**	Current indent.
6	**.l**	Current line length.
4	**.n**	Length of text portion on previous output line.
3	**.o**	Current page offset.
3	**.p**	Current page length.
2.3	**.s**	Current point size.
7.5	**.t**	Distance to the next trap.
4.1	**.u**	Equal to 1 in fill mode and 0 in nofill mode.
5.1	**.v**	Current vertical line spacing.
11.2	**.w**	Width of previous character.
-	**.x**	Reserved version-dependent register.
-	**.y**	Reserved version-dependent register.
7.4	**.z**	Name of current diversion.

REFERENCE MANUAL

1. General Explanation

1.1. Form of input. Input consists of *text lines*, which are destined to be printed, interspersed with *control lines*, which set parameters or otherwise control subsequent processing. Control lines begin with a *control character*—normally . (period) or ´ (acute accent)—followed by a one or two character name that specifies a basic *request* or the substitution of a user-defined *macro* in place of the control line. The control character ´ suppresses the *break* function—the forced output of a partially filled line—caused by certain requests. The control character may be separated from the request/macro name by white space (spaces and/or tabs) for esthetic reasons. Names must be followed by either space or newline. Control lines with unrecognized names are ignored.

Various special functions may be introduced anywhere in the input by means of an *escape* character, normally \. For example, the function **\n***R* causes the interpolation of the contents of the *number register R* in place of the function; here *R* is either a single character name as in **\n***x*, or left-parenthesis-introduced, two-character name as in **\n(***xx*.

1.2. Formatter and device resolution. TROFF internally uses 432 units/inch, corresponding to the Graphic Systems phototypesetter which has a horizontal resolution of 1/432 inch and a vertical resolution of 1/144 inch. NROFF internally uses 240 units/inch, corresponding to the least common multiple of the horizontal and vertical resolutions of various typewriter-like output devices. TROFF rounds horizontal/vertical numerical parameter input to the actual horizontal/vertical resolution of the Graphic Systems typesetter. NROFF similarly rounds numerical input to the actual resolution of the output device indicated by the **−T** option (default Model 37 Teletype).

1.3. Numerical parameter input. Both NROFF and TROFF accept numerical input with the appended scale indicators shown in the following table, where *S* is the current type size in points, *V* is the current vertical line spacing in basic units, and *C* is a *nominal character width* in basic units.

Scale Indicator	Meaning	Number of basic units TROFF	NROFF
i	Inch	432	240
c	Centimeter	432×50/127	240×50/127
P	Pica = 1/6 inch	72	240/6
m	Em = *S* points	6×*S*	*C*
n	En = Em/2	3×*S*	*C, same as Em*
p	Point = 1/72 inch	6	240/72
u	Basic unit	1	1
v	Vertical line space	*V*	*V*
none	Default, see below		

In NROFF, *both* the em and the en are taken to be equal to the *C*, which is output-device dependent; common values are 1/10 and 1/12 inch. Actual character widths in NROFF need not be all the same and constructed characters such as −> (→) are often extra wide. The default scaling is ems for the horizontally-oriented requests and functions **ll**, **in**, **ti**, **ta**, **lt**, **po**, **mc**, **\h**, and **\l**; *V*s for the vertically-oriented requests and functions **pl**, **wh**, **ch**, **dt**, **sp**, **sv**, **ne**, **rt**, **\v**, **\x**, and **\L**; **p** for the **vs** request; and **u** for the requests **nr**, **if**, and **ie**. *All* other requests ignore any scale indicators. When a number register containing an already appropriately scaled number is interpolated to provide numerical input, the unit scale indicator **u** may need to be appended to prevent an additional inappropriate default scaling.

The number, *N*, may be specified in decimal-fraction form but the parameter finally stored is rounded to an integer number of basic units.

The *absolute position* indicator | may be prepended to a number *N* to generate the distance to the vertical or horizontal place *N*. For vertically-oriented requests and functions, |*N* becomes the distance in basic units from the current vertical place on the page or in a *diversion* (§7.4) to the the vertical place *N*. For *all* other requests and functions, |*N* becomes the distance from the current horizontal place on the *input* line to the horizontal place *N*. For example,

.sp |3.2c

will space *in the required direction* to 3.2 centimeters from the top of the page.

1.4. Numerical expressions. Wherever numerical input is expected an expression involving parentheses, the arithmetic operators +, −, /, *, % (mod), and the logical operators <, >, <=, >=, = (or ==), & (and), : (or) may be used. Except where controlled by parentheses, evaluation of expressions is left-to-right; there is no operator precedence. In the case of certain requests, an initial + or − is stripped and interpreted as an increment or decrement indicator respectively. In the presence of default scaling, the desired scale indicator must be attached to *every* number in an expression for which the desired and default scaling differ. For example, if the number register **x** contains 2 and the current point size is 10, then

.ll (4.25i+\nxP+3)/2u

will set the line length to 1/2 the sum of 4.25 inches + 2 picas + 30 points.

1.5. Notation. Numerical parameters are indicated in this manual in two ways. ±*N* means that the argument may take the forms *N*, +*N*, or −*N* and that the corresponding effect is to set the affected parameter to *N*, to increment it by *N*, or to decrement it by *N* respectively. Plain *N* means that an initial algebraic sign is *not* an increment indicator, but merely the sign of *N*. Generally, unreasonable numerical input is either ignored or truncated to a reasonable value. For example, most requests expect to set parameters to non-negative values; exceptions are **sp**, **wh**, **ch**, **nr**, and **if**. The requests **ps**, **ft**, **po**, **vs**, **ls**, **ll**, **in**, and **lt** restore the *previous* parameter value in the *absence* of an argument.

Single character arguments are indicated by single lower case letters and one/two character arguments are indicated by a pair of lower case letters. Character string arguments are indicated by multi-character mnemonics.

2. Font and Character Size Control

2.1. Character set. The TROFF character set consists of the Graphics Systems Commercial II character set plus a Special Mathematical Font character set—each having 102 characters. These character sets are shown in the attached Table I. All ASCII characters are included, with some on the Special Font. With three exceptions, the ASCII characters are input as themselves, and non-ASCII characters are input in the form \\(*xx* where *xx* is a two-character name given in the attached Table II. The three ASCII exceptions are mapped as follows:

ASCII Input		Printed by TROFF	
Character	Name	Character	Name
´	acute accent	’	close quote
`	grave accent	‘	open quote
−	minus	-	hyphen

The characters ´, `, and − may be input by \´, \`, and \− respectively or by their names (Table II). The ASCII characters @, #, ", ´, `, <, >, \, {, }, ~, ^, and _ exist only on the Special Font and are printed as a 1-em space if that Font is not mounted.

NROFF understands the entire TROFF character set, but can in general print only ASCII characters, additional characters as may be available on the output device, such characters as may be able to be constructed by overstriking or other combination, and those that can reasonably be mapped into other printable characters. The exact behavior is determined by a driving table prepared for each device. The characters ´, `, and _ print as themselves.

2.2. Fonts. The default mounted fonts are Times Roman (**R**), Times Italic (**I**), Times Bold (**B**), and the Special Mathematical Font (**S**) on physical typesetter positions 1, 2, 3, and 4 respectively. These fonts are used in this document. The *current* font, initially Roman, may be changed (among the mounted fonts) by use of the **ft** request, or by imbedding at any desired point either **\f***x*, **\f(***xx*, or **\f***N* where *x* and *xx* are the name of a mounted font and *N* is a numerical font position. It is *not* necessary to change to the Special font; characters on that font are automatically handled. A request for a named but not-mounted font is *ignored.* TROFF can be informed that any particular font is mounted by use of the **fp** request. The list of known fonts is installation dependent. In the subsequent discussion of font-related requests, *F* represents either a one/two-character font name or the numerical font position, 1-4. The current font is available (as numerical position) in the read-only number register **.f**.

NROFF understands font control and normally underlines Italic characters (see §10.5).

2.3. Character size. Character point sizes available on the Graphic Systems typesetter are 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36. This is a range of 1/12 inch to 1/2 inch. The **ps** request is used to change or restore the point size. Alternatively the point size may be changed between any two characters by imbedding a **\s***N* at the desired point to set the size to *N*, or a **\s**±*N* ($1 \leqslant N \leqslant 9$) to increment/decrement the size by *N*; **\s0** restores the *previous* size. Requested point size values that are between two valid sizes yield the larger of the two. The current size is available in the **.s** register. NROFF ignores type size control.

Request Form	***Initial Value***	***If No Argument***	***Notes****	***Explanation***
.ps ±*N*	10 point	previous	E	Point size set to ±*N*. Alternatively imbed **\s***N* or **\s**±*N*. Any positive size value may be requested; if invalid, the next larger valid size will result, with a maximum of 36. A paired sequence +*N*, −*N* will work because the previous requested value is also remembered. Ignored in NROFF.
.ss *N*	12/36 em	ignored	E	Space-character size is set to *N*/36 ems. This size is the minimum word spacing in adjusted text. Ignored in NROFF.
.cs *F N M*	off	-	P	Constant character space (width) mode is set on for font *F* (if mounted); the width of every character will be taken to be *N*/36 ems. If *M* is absent, the em is that of the character's point size; if *M* is given, the em is *M*-points. All affected characters are centered in this space, including those with an actual width larger than this space. Special Font characters occurring while the current font is *F* are also so treated. If *N* is absent, the mode is turned off. The mode must be still or again in effect when the characters are physically printed. Ignored in NROFF.
.bd *F N*	off	-	P	The characters in font *F* will be artificially emboldened by printing each one twice, separated by *N*−1 basic units. A reasonable value for *N* is 3 when the character size is in the vicinity of 10 points. If *N* is missing the embolden mode is turned off. The column heads above were printed with **.bd I 3**. The mode must be still or again in effect when the characters are physically printed. Ignored in NROFF.

*Notes are explained at the end of the Summary and Index above.

.bd S *F N*	off	-	P	The characters in the Special Font will be emboldened whenever the current font is *F*. This manual was printed with **.bd** S B 3. The mode must be still or again in effect when the characters are physically printed.
.ft *F*	Roman	previous	E	Font changed to *F*. Alternatively, imbed **\f***F*. The font name **P** is reserved to mean the previous font.
.fp *N F*	R,I,B,S	ignored	-	Font position. This is a statement that a font named *F* is mounted on position *N* (1-4). It is a fatal error if *F* is not known. The phototypesetter has four fonts physically mounted. Each font consists of a film strip which can be mounted on a numbered quadrant of a wheel. The default mounting sequence assumed by TROFF is R, I, B, and S on positions 1, 2, 3 and 4.

3. Page control

Top and bottom margins are *not* automatically provided; it is conventional to define two *macros* and to set *traps* for them at vertical positions 0 (top) and $-N$ (*N* from the bottom). See §7 and Tutorial Examples §T2. A pseudo-page transition onto the *first* page occurs either when the first *break* occurs or when the first *non-diverted* text processing occurs. Arrangements for a trap to occur at the top of the first page must be completed before this transition. In the following, references to the *current diversion* (§7.4) mean that the mechanism being described works during both ordinary and diverted output (the former considered as the top diversion level).

The useable page width on the Graphic Systems phototypesetter is about 7.54 inches, beginning about 1/27 inch from the left edge of the 8 inch wide, continuous roll paper. The physical limitations on NROFF output are output-device dependent.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.pl $\pm N$	11 in	11 in	v	Page length set to $\pm N$. The internal limitation is about 75 inches in TROFF and about 136 inches in NROFF. The current page length is available in the **.p** register.
.bp $\pm N$	*N*=1	-	B*,v	Begin page. The current page is ejected and a new page is begun. If $\pm N$ is given, the new page number will be $\pm N$. Also see request **ns**.
.pn $\pm N$	*N*=1	ignored	-	Page number. The next page (when it occurs) will have the page number $\pm N$. A **pn** must occur before the initial pseudo-page transition to effect the page number of the first page. The current page number is in the **%** register.
.po $\pm N$	0; 26/27 in†	previous	v	Page offset. The current *left margin* is set to $\pm N$. The TROFF initial value provides about 1 inch of paper margin including the physical typesetter margin of 1/27 inch. In TROFF the maximum (line-length)+(page-offset) is about 7.54 inches. See §6. The current page offset is available in the **.o** register.
.ne *N*	-	*N*=1 *V*	D,v	Need *N* vertical space. If the distance, *D*, to the next trap position (see §7.5) is less than *N*, a forward vertical space of size *D* occurs, which will spring the trap. If there are no remaining traps on the page, *D* is the

*The use of " ' " as control character (instead of ".") suppresses the break function.

†Values separated by ";" are for NROFF and TROFF respectively.

				distance to the bottom of the page. If $D< V$, another line could still be output and spring the trap. In a diversion, D is the distance to the *diversion trap*, if any, or is very large.
.mk *R*	none	internal	D	Mark the *current* vertical place in an internal register (both associated with the current diversion level), or in register *R*, if given. See **rt** request.
.rt ±*N*	none	internal	D,v	Return *upward only* to a marked vertical place in the current diversion. If ±*N* (w.r.t. current place) is given, the place is ±*N* from the top of the page or diversion or, if *N* is absent, to a place marked by a previous **mk**. Note that the **sp** request (§5.3) may be used in all cases instead of **rt** by spacing to the absolute place stored in a explicit register; e. g. using the sequence **.mk** *R* ... **.sp** |**n***R***u**.

4. Text Filling, Adjusting, and Centering

4.1. Filling and adjusting. Normally, words are collected from input text lines and assembled into a output text line until some word doesn't fit. An attempt is then made the hyphenate the word in effort to assemble a part of it into the output line. The spaces between the words on the output line are then increased to spread out the line to the current *line length* minus any current *indent.* A *word* is any string of characters delimited by the *space* character or the beginning/end of the input line. Any adjacent pair of words that must be kept together (neither split across output lines nor spread apart in the adjustment process) can be tied together by separating them with the *unpaddable space* character "\ " (backslash-space). The adjusted word spacings are uniform in TROFF and the minimum interword spacing can be controlled with the **ss** request (§2). In NROFF, they are normally nonuniform because of quantization to character-size spaces; however, the command line option **−e** causes uniform spacing with full output device resolution. Filling, adjustment, and hyphenation (§13) can all be prevented or controlled. The *text length* on the last line output is available in the **.n** register, and text base-line position on the page for this line is in the **nl** register. The text base-line high-water mark (lowest place) on the current page is in the **.h** register.

An input text line ending with **.**, **?**, or **!** is taken to be the end of a *sentence*, and an additional space character is automatically provided during filling. Multiple inter-word space characters found in the input are retained, except for trailing spaces; initial spaces also cause a *break.*

When filling is in effect, a **\p** may be imbedded or attached to a word to cause a *break* at the *end* of the word and have the resulting output line *spread out* to fill the current line length.

A text input line that happens to begin with a control character can be made to not look like a control line by prefacing it with the non-printing, zero-width filler character **\&**. Still another way is to specify output translation of some convenient character into the control character using **tr** (§10.5).

4.2. Interrupted text. The copying of a input line in *nofill* (non-fill) mode can be *interrupted* by terminating the partial line with a **\c**. The *next* encountered input text line will be considered to be a continuation of the same line of input text. Similarly, a word within *filled* text may be interrupted by terminating the word (and line) with **\c**; the next encountered text will be taken as a continuation of the interrupted word. If the intervening control lines cause a break, any partial line will be forced out along with any partial word.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.br	-	-	B	Break. The filling of the line currently being collected is stopped and the line is output without adjustment. Text lines beginning with space characters and empty text lines (blank lines) also cause a break.

.fi	fill on	-	B,E	Fill subsequent output lines. The register **.u** is 1 in fill mode and 0 in nofill mode.
.nf	fill on	-	B,E	Nofill. Subsequent output lines are *neither* filled *nor* adjusted. Input text lines are copied directly to output lines *without regard* for the current line length.
.ad *c*	adj,both	adjust	E	Line adjustment is begun. If fill mode is not on, adjustment will be deferred until fill mode is back on. If the type indicator *c* is present, the adjustment type is changed as shown in the following table.

Indicator	Adjust Type
l	adjust left margin only
r	adjust right margin only
c	center
b or **n**	adjust both margins
absent	unchanged

.na	adjust	-	E	Noadjust. Adjustment is turned off; the right margin will be ragged. The adjustment type for **ad** is not changed. Output line filling still occurs if fill mode is on.
.ce *N*	off	*N*=1	B,E	Center the next *N* input text lines within the current (line-length minus indent). If *N*=0, any residual count is cleared. A break occurs after each of the *N* input lines. If the input line is too long, it will be left adjusted.

5. Vertical Spacing

*5.1. **Base-line spacing.*** The vertical spacing *(V)* between the base-lines of successive output lines can be set using the **vs** request with a resolution of 1/144inch = 1/2 point in TROFF, and to the output device resolution in NROFF. *V* must be large enough to accommodate the character sizes on the affected output lines. For the common type sizes (9-12 points), usual typesetting practice is to set *V* to 2 points greater than the point size; TROFF default is 10-point type on a 12-point spacing (as in this document). The current *V* is available in the **.v** register. Multiple-*V* line separation (e.g. double spacing) may be requested with **ls**.

*5.2. **Extra line-space.*** If a word contains a vertically tall construct requiring the output line containing it to have extra vertical space before and/or after it, the *extra-line-space* function **\x**'*N*' can be imbedded in or attached to that word. In this and other functions having a pair of delimiters around their parameter (here '), the delimiter choice is arbitrary, except that it can't look like the continuation of a number expression for *N*. If *N* is negative, the output line containing the word will be preceded by *N* extra vertical space; if *N* is positive, the output line containing the word will be followed by *N* extra vertical space. If successive requests for extra space apply to the same line, the maximum values are used. The most recently utilized post-line extra line-space is available in the **.a** register.

*5.3. **Blocks of vertical space.*** A block of vertical space is ordinarily requested using **sp**, which honors the *no-space* mode and which does not space *past* a trap. A contiguous block of vertical space may be reserved using **sv**.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.vs *N*	1/6in;12pts	previous	E,p	Set vertical base-line spacing size *V*. Transient *extra* vertical space available with **\x**'*N*' (see above).
.ls *N*	*N*=1	previous	E	*Line* spacing set to ±*N*. *N*−1 *V*s *(blank lines)* are appended to each output text line. Appended blank lines are omitted, if the text or previous appended blank line

				reached a trap position.
.sp *N*	-	*N*=1*V*	B,v	Space vertically in *either* direction. If *N* is negative, the motion is *backward* (upward) and is limited to the distance to the top of the page. Forward (downward) motion is truncated to the distance to the nearest trap. If the no-space mode is on, no spacing occurs (see **ns**, and **rs** below).
.sv *N*	-	*N*=1*V*	v	Save a contiguous vertical block of size *N*. If the distance to the next trap is greater than *N*, *N* vertical space is output. No-space mode has *no* effect. If this distance is less than *N*, no vertical space is immediately output, but *N* is remembered for later output (see **os**). Subsequent **sv** requests will overwrite any still remembered *N*.
.os	-	-	-	Output saved vertical space. No-space mode has *no* effect. Used to finally output a block of vertical space requested by an earlier **sv** request.
.ns	space	-	D	No-space mode turned on. When on, the no-space mode inhibits **sp** requests and **bp** requests *without* a next page number. The no-space mode is turned off when a line of output occurs, or with **rs**.
.rs	space	-	D	Restore spacing. The no-space mode is turned off.
Blank text line.		-	B	Causes a break and output of a blank line exactly like **sp 1**.

6. Line Length and Indenting

The maximum line length for fill mode may be set with **ll**. The indent may be set with **in**; an indent applicable to *only* the *next* output line may be set with **ti**. The line length includes indent space but *not* page offset space. The line-length minus the indent is the basis for centering with **ce**. The effect of **ll**, **in**, or **ti** is delayed, if a partially collected line exists, until after that line is output. In fill mode the length of text on an output line is less than or equal to the line length minus the indent. The current line length and indent are available in registers **.l** and **.i** respectively. The length of *three-part titles* produced by **tl** (see §14) is *independently* set by **lt**.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.ll ±*N*	6.5 in	previous	E,m	Line length is set to ±*N*. In TROFF the maximum (line-length)+(page-offset) is about 7.54 inches.
.in ±*N*	*N*=0	previous	B,E,m	Indent is set to ±*N*. The indent is prepended to each output line.
.ti ±*N*	-	ignored	B,E,m	Temporary indent. The *next* output text line will be indented a distance ±*N* with respect to the current indent. The resulting total indent may not be negative. The current indent is not changed.

7. Macros, Strings, Diversion, and Position Traps

7.1. Macros and strings. A *macro* is a named set of arbitrary *lines* that may be invoked by name or with a *trap*. A *string* is a named string of *characters*, *not* including a newline character, that may be interpolated by name at any point. Request, macro, and string names share the *same* name list. Macro and string names may be one or two characters long and may usurp previously defined request, macro, or string names. Any of these entities may be renamed with **rn** or removed with **rm**. Macros are created by **de** and **di**, and appended to by **am** and **da**; **di** and **da** cause normal output to be stored in a macro. Strings are created by **ds** and appended to by **as**. A macro is invoked in the same way as a request; a control line beginning .*xx* will interpolate the contents of macro *xx*. The remainder of the line may contain up to nine *arguments*. The strings *x* and *xx* are interpolated at any desired point with \\•*x* and \\•(*xx* respectively. String references and macro invocations may be nested.

7.2. Copy mode input interpretation. During the definition and extension of strings and macros (not by diversion) the input is read in *copy mode*. The input is copied without interpretation *except* that:

- The contents of number registers indicated by **\n** are interpolated.
- Strings indicated by **\•** are interpolated.
- Arguments indicated by **\$** are interpolated.
- Concealed newlines indicated by \\(newline) are eliminated.
- Comments indicated by **\"** are eliminated.
- **\t** and **\a** are interpreted as ASCII horizontal tab and SOH respectively (§9).
- \\\\ is interpreted as \\.
- \\. is interpreted as ".".

These interpretations can be suppressed by prepending a \\. For example, since \\\\ maps into a \\, \\\n will copy as **\n** which will be interpreted as a number register indicator when the macro or string is reread.

7.3. Arguments. When a macro is invoked by name, the remainder of the line is taken to contain up to nine arguments. The argument separator is the space character, and arguments may be surrounded by double-quotes to permit imbedded space characters. Pairs of double-quotes may be imbedded in double-quoted arguments to represent a single double-quote. If the desired arguments won't fit on a line, a concealed newline may be used to continue on the next line.

When a macro is invoked the *input level* is *pushed down* and any arguments available at the previous level become unavailable until the macro is completely read and the previous level is restored. A macro's own arguments can be interpolated at *any* point within the macro with **\$***N*, which interpolates the *N*th argument ($1 \leqslant N \leqslant 9$). If an invoked argument doesn't exist, a null string results. For example, the macro *xx* may be defined by

```
.de xx      \"begin definition
Today is \\$1 the \\$2.
..          \"end definition
```

and called by

```
.xx Monday 14th
```

to produce the text

Today is Monday the 14th.

Note that the **\$** was concealed in the definition with a prepended \. The number of currently available arguments is in the **.$** register.

No arguments are available at the top (non-macro) level in this implementation. Because string referencing is implemented as a input-level push down, no arguments are available from *within* a string. No arguments are available within a trap-invoked macro.

Arguments are copied in *copy mode* onto a stack where they are available for reference. The mechanism does not allow an argument to contain a direct reference to a *long* string (interpolated at copy time) and it is advisable to conceal string references (with an extra \\) to delay interpolation until argument reference time.

7.4. Diversions. Processed output may be diverted into a macro for purposes such as footnote processing (see Tutorial §T5) or determining the horizontal and vertical size of some text for conditional changing of pages or columns. A single diversion trap may be set at a specified vertical position. The number registers **dn** and **dl** respectively contain the vertical and horizontal size of the most recently ended diversion. Processed text that is diverted into a macro retains the vertical size of each of its lines when reread in *nofill* mode regardless of the current *V*. Constant-spaced (**cs**) or emboldened (**bd**) text that is diverted can be reread correctly only if these modes are again or still in effect at reread time. One way to do this is to imbed in the diversion the appropriate **cs** or **bd** requests with the *transparent* mechanism described in §10.6.

Diversions may be nested and certain parameters and registers are associated with the current diversion level (the top non-diversion level may be thought of as the 0th diversion level). These are the diversion trap and associated macro, no-space mode, the internally-saved marked place (see **mk** and **rt**), the current vertical place (**.d** register), the current high-water text base-line (**.h** register), and the current diversion name (**.z** register).

7.5. Traps. Three types of trap mechanisms are available—page traps, a diversion trap, and an input-line-count trap. Macro-invocation traps may be planted using **wh** at any page position including the top. This trap position may be changed using **ch**. Trap positions at or below the bottom of the page have no effect unless or until moved to within the page or rendered effective by an increase in page length. Two traps may be planted at the *same* position only by first planting them at different positions and then moving one of the traps; the first planted trap will conceal the second unless and until the first one is moved (see Tutorial Examples §T5). If the first one is moved back, it again conceals the second trap. The macro associated with a page trap is automatically invoked when a line of text is output whose vertical size *reaches* or *sweeps past* the trap position. Reaching the bottom of a page springs the top-of-page trap, if any, provided there is a next page. The distance to the next trap position is available in the **.t** register; if there are no traps between the current position and the bottom of the page, the distance returned is the distance to the page bottom.

A macro-invocation trap effective in the current diversion may be planted using **dt**. The **.t** register works in a diversion; if there is no subsequent trap a *large* distance is returned. For a description of input-line-count traps, see **it** below.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.de *xx yy*	-	*.yy*=..	-	Define or redefine the macro *xx*. The contents of the macro begin on the next input line. Input lines are copied in *copy mode* until the definition is terminated by a line beginning with *.yy*, whereupon the macro *yy* is called. In the absence of *yy*, the definition is terminated by a line beginning with "..". A macro may contain **de** requests provided the terminating macros differ or the contained definition terminator is concealed. ".." can be concealed as \\.. which will copy as \.. and be reread as "..".
.am *xx yy*	-	*.yy*=..	-	Append to macro (append version of **de**).
.ds *xx string*	-	ignored	-	Define a string *xx* containing *string*. Any initial double-quote in *string* is stripped off to permit initial blanks.
.as *xx string*	-	ignored	-	Append *string* to string *xx* (append version of **ds**).
.rm *xx*	-	ignored	-	Remove request, macro, or string. The name *xx* is removed from the name list and any related storage space is freed. Subsequent references will have no effect.
.rn *xx yy*	-	ignored	-	Rename request, macro, or string *xx* to *yy*. If *yy* exists, it is first removed.
.di *xx*	-	end	D	Divert output to macro *xx*. Normal text processing occurs during diversion except that page offsetting is not done. The diversion ends when the request **di** or **da** is encountered without an argument; extraneous requests of this type should not appear when nested diversions are being used.

.da *xx*	-	end	D	Divert, appending to *xx* (append version of **di**).
.wh *N xx*	-	-	v	Install a trap to invoke *xx* at page position *N;* a *negative N* will be interpreted with respect to the page *bottom.* Any macro previously planted at *N* is replaced by *xx.* A zero *N* refers to the *top* of a page. In the absence of *xx*, the first found trap at *N*, if any, is removed.
.ch *xx N*	-	-	v	Change the trap position for macro *xx* to be *N.* In the absence of *N*, the trap, if any, is removed.
.dt *N xx*	-	off	D,v	Install a diversion trap at position *N* in the *current* diversion to invoke macro *xx.* Another **dt** will redefine the diversion trap. If no arguments are given, the diversion trap is removed.
.it *N xx*	-	off	E	Set an input-line-count trap to invoke the macro *xx* after *N* lines of *text* input have been read (control or request lines don't count). The text may be in-line text or text interpolated by inline or trap-invoked macros.
.em *xx*	none	none	-	The macro *xx* will be invoked when all input has ended. The effect is the same as if the contents of *xx* had been at the end of the last file processed.

8. Number Registers

A variety of parameters are available to the user as predefined, named *number registers* (see Summary and Index, page 7). In addition, the user may define his own named registers. Register names are one or two characters long and *do not* conflict with request, macro, or string names. Except for certain predefined read-only registers, a number register can be read, written, automatically incremented or decremented, and interpolated into the input in a variety of formats. One common use of user-defined registers is to automatically number sections, paragraphs, lines, etc. A number register may be used any time numerical input is expected or desired and may be used in numerical *expressions* (§1.4).

Number registers are created and modified using **nr**, which specifies the name, numerical value, and the auto-increment size. Registers are also modified, if accessed with an auto-incrementing sequence. If the registers *x* and *xx* both contain *N* and have the auto-increment size *M*, the following access sequences have the effect shown:

Sequence	Effect on Register	Value Interpolated
\n*x*	none	*N*
\n(*xx*	none	*N*
\n+*x*	*x* incremented by *M*	*N+M*
\n−*x*	*x* decremented by *M*	*N−M*
\n+(*xx*	*xx* incremented by *M*	*N+M*
\n−(*xx*	*xx* decremented by *M*	*N−M*

When interpolated, a number register is converted to decimal (default), decimal with leading zeros, lower-case Roman, upper-case Roman, lower-case sequential alphabetic, or upper-case sequential alphabetic according to the format specified by **af**.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.nr *R ±N M*		-	u	The number register *R* is assigned the value ±*N* with respect to the previous value, if any. The increment for auto-incrementing is set to *M.*

.af *R c*	arabic	-	-	Assign format *c* to register *R*. The available formats are:

Format	Numbering Sequence
1	0,1,2,3,4,5,...
001	000,001,002,003,004,005,...
i	0,i,ii,iii,iv,v,...
I	0,I,II,III,IV,V,...
a	0,a,b,c,...,z,aa,ab,...,zz,aaa,...
A	0,A,B,C,...,Z,AA,AB,...,ZZ,AAA,...

An arabic format having *N* digits specifies a field width of *N* digits (example 2 above). The read-only registers and the *width* function (§11.2) are always arabic.

.rr *R*	-	ignored	-	Remove register *R*. If many registers are being created dynamically, it may become necessary to remove no longer used registers to recapture internal storage space for newer registers.

9. Tabs, Leaders, and Fields

9.1. Tabs and leaders. The ASCII horizontal tab character and the ASCII SOH (hereafter known as the *leader* character) can both be used to generate either horizontal motion or a string of repeated characters. The length of the generated entity is governed by internal *tab stops* specifiable with **ta**. The default difference is that tabs generate motion and leaders generate a string of periods; **tc** and **lc** offer the choice of repeated character or motion. There are three types of internal tab stops—*left* adjusting, *right* adjusting, and *centering*. In the following table: *D* is the distance from the current position on the *input* line (where a tab or leader was found) to the next tab stop; *next-string* consists of the input characters following the tab (or leader) up to the next tab (or leader) or end of line; and *W* is the width of *next-string*.

Tab type	Length of motion or repeated characters	Location of *next-string*
Left	*D*	Following *D*
Right	$D-W$	Right adjusted within *D*
Centered	$D-W/2$	Centered on right end of *D*

The length of generated motion is allowed to be negative, but that of a repeated character string cannot be. Repeated character strings contain an integer number of characters, and any residual distance is prepended as motion. Tabs or leaders found after the last tab stop are ignored, but may be used as *next-string* terminators.

Tabs and leaders are not interpreted in *copy mode.* **\t** and **\a** always generate a non-interpreted tab and leader respectively, and are equivalent to actual tabs and leaders in *copy mode.*

9.2. Fields. A *field* is contained between a *pair* of *field delimiter* characters, and consists of sub-strings separated by *padding* indicator characters. The field length is the distance on the *input* line from the position where the field begins to the next tab stop. The difference between the total length of all the sub-strings and the field length is incorporated as horizontal padding space that is divided among the indicated padding places. The incorporated padding is allowed to be negative. For example, if the field delimiter is **#** and the padding indicator is **^**, **#^***xxx***^***right***#** specifies a right-adjusted string with the string *xxx* centered in the remaining space.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.ta *Nt* ...	0.8; 0.5in	none	E,m	Set tab stops and types. *t*=**R**, right adjusting; *t*=**C**, centering; *t* absent, left adjusting. TROFF tab stops are preset every 0.5in.; NROFF every 0.8in. The stop values are separated by spaces, and a value preceded by **+** is treated as an increment to the previous stop value.
.tc *c*	none	none	E	The tab repetition character becomes *c*, or is removed specifying motion.
.lc *c*	**.**	none	E	The leader repetition character becomes *c*, or is removed specifying motion.
.fc *a b*	off	off	-	The field delimiter is set to *a*; the padding indicator is set to the *space* character or to *b*, if given. In the absence of arguments the field mechanism is turned off.

10. Input and Output Conventions and Character Translations

10.1. Input character translations. Ways of inputting the graphic character set were discussed in §2.1. The ASCII control characters horizontal tab (§9.1), SOH (§9.1), and backspace (§10.3) are discussed elsewhere. The newline delimits input lines. In addition, STX, ETX, ENQ, ACK, and BEL are accepted, and may be used as delimiters or translated into a graphic with **tr** (§10.5). *All* others are ignored.

The *escape* character \ introduces *escape sequences*—causes the following character to mean another character, or to indicate some function. A complete list of such sequences is given in the Summary and Index on page 6. \ should not be confused with the ASCII control character ESC of the same name. The escape character \ can be input with the sequence \\. The escape character can be changed with **ec**, and all that has been said about the default \ becomes true for the new escape character. **\e** can be used to print whatever the current escape character is. If necessary or convenient, the escape mechanism may be turned off with **eo**, and restored with **ec**.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.ec *c*	\	\	-	Set escape character to \, or to *c*, if given.
.eo	on	-	-	Turn escape mechanism off.

10.2. Ligatures. Five ligatures are available in the current TROFF character set — **ﬁ**, **ﬂ**, **ﬀ**, **ﬃ**, and **ﬄ**. They may be input (even in NROFF) by **\(fi**, **\(fl**, **\(ff**, **\(Fi**, and **\(Fl** respectively. The ligature mode is normally on in TROFF, and *automatically* invokes ligatures during input.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.lg *N*	off; on	on	-	Ligature mode is turned on if *N* is absent or non-zero, and turned off if *N*=0. If *N*=2, only the two-character ligatures are automatically invoked. Ligature mode is inhibited for request, macro, string, register, or file names, and in *copy mode*. No effect in NROFF.

10.3. Backspacing, underlining, overstriking, etc. Unless in *copy mode*, the ASCII backspace character is replaced by a backward horizontal motion having the width of the space character. Underlining as a form of line-drawing is discussed in §12.4. A generalized overstriking function is described in §12.1.

NROFF automatically underlines characters in the *underline* font, specifiable with **uf**, normally that on font position 2 (normally Times Italic, see §2.2). In addition to **ft** and **\f***F*, the underline font may be selected by **ul** and **cu**. Underlining is restricted to an output-device-dependent subset of *reasonable* characters.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.ul *N*	off	*N*=1	E	Underline in NROFF (italicize in TROFF) the next *N* input text lines. Actually, switch to *underline* font, saving the current font for later restoration; *other* font changes within the span of a **ul** will take effect, but the restoration will undo the last change. Output generated by **tl** (§14) *is* affected by the font change, but does *not* decrement *N*. If *N*> 1, there is the risk that a trap interpolated macro may provide text lines within the span; environment switching can prevent this.
.cu *N*	off	*N*=1	E	A variant of **ul** that causes *every* character to be underlined in NROFF. Identical to **ul** in TROFF.
.uf *F*	Italic	Italic	-	Underline font set to *F*. In NROFF, *F* may *not* be on position 1 (initially Times Roman).

10.4. Control characters. Both the control character . and the *no-break* control character ' may be changed, if desired. Such a change must be compatible with the design of any macros used in the span of the change, and particularly of any trap-invoked macros.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.cc *c*	.	.	E	The basic control character is set to *c*, or reset to ".".
.c2 *c*	'	'	E	The *nobreak* control character is set to *c*, or reset to "'".

10.5. Output translation. One character can be made a stand-in for another character using **tr**. All text processing (e. g. character comparisons) takes place with the input (stand-in) character which appears to have the width of the final character. The graphic translation occurs at the moment of output (including diversion).

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.tr *abcd....*	none	-	O	Translate *a* into *b*, *c* into *d*, etc. If an odd number of characters is given, the last one will be mapped into the space character. To be consistent, a particular translation must stay in effect from *input* to *output* time.

10.6. Transparent throughput. An input line beginning with a \! is read in *copy mode* and *transparently* output (without the initial \!); the text processor is otherwise unaware of the line's presence. This mechanism may be used to pass control information to a post-processor or to imbed control lines in a macro created by a diversion.

10.7. Comments and concealed newlines. An uncomfortably long input line that must stay one line (e. g. a string definition, or nofilled text) can be split into many physical lines by ending all but the last one with the escape \. The sequence \(newline) is *always* ignored—except in a comment. Comments may be imbedded at the *end* of any line by prefacing them with \". The newline at the end of a comment cannot be concealed. A line beginning with \" will appear as a blank line and behave like **.sp 1**; a comment can be on a line by itself by beginning the line with .\".

11. Local Horizontal and Vertical Motions, and the Width Function

11.1. Local Motions. The functions **\v**'*N*' and **\h**'*N*' can be used for *local* vertical and horizontal motion respectively. The distance *N* may be negative; the *positive* directions are *rightward* and *downward*. A *local* motion is one contained *within* a line. To avoid unexpected vertical dislocations, it is necessary that the *net* vertical local motion within a word in filled text and otherwise within a line balance to zero. The above and certain other escape sequences providing local motion are summarized in the following table.

Vertical Local Motion	Effect in TROFF	NROFF	Horizontal Local Motion	Effect in TROFF	NROFF
\v'*N*'	Move distance *N*		**\h**'*N*'	Move distance *N*	
			\(space)	Unpaddable space-size space	
\u	½ em up	½ line up	**\0**	Digit-size space	
\d	½ em down	½ line down			
\r	1 em up	1 line up	**\\|**	1/6 em space	ignored
			\^	1/12 em space	ignored

As an example, $\mathbf{E^2}$ could be generated by the sequence **E\s−2\v'−0.4m'2\v'0.4m'\s+2**; it should be noted in this example that the 0.4 em vertical motions are at the smaller size.

11.2. Width Function. The *width* function **\w**'*string*' generates the numerical width of *string* (in basic units). Size and font changes may be safely imbedded in *string*, and will not affect the current environment. For example, **.ti −\w'1. 'u** could be used to temporarily indent leftward a distance equal to the size of the string "**1.** ".

The width function also sets three number registers. The registers **st** and **sb** are set respectively to the highest and lowest extent of *string* relative to the baseline; then, for example, the total *height* of the string is **\n(stu−\n(sbu**. In TROFF the number register **ct** is set to a value between 0 and 3: 0 means that all of the characters in *string* were short lower case characters without descenders (like **e**); 1 means that at least one character has a descender (like **y**); 2 means that at least one character is tall (like **H**); and 3 means that both tall characters and characters with descenders are present.

11.3. Mark horizontal place. The escape sequence **\k***x* will cause the *current* horizontal position in the *input line* to be stored in register *x*. As an example, the construction **\kx***word***\h'|\nxu+2u'***word* will embolden *word* by backing up to almost its beginning and overprinting it, resulting in ***word***.

12. Overstrike, Bracket, Line-drawing, and Zero-width Functions

12.1. Overstriking. Automatically centered overstriking of up to nine characters is provided by the *overstrike* function **\o'***string***'**. The characters in *string* overprinted with centers aligned; the total width is that of the widest character. *string* should *not* contain local vertical motion. As examples, **\o'e\''** produces **é**, and **\o'\(mo\(sl'** produces **∉**.

12.2. Zero-width characters. The function **\z***c* will output *c* without spacing over it, and can be used to produce left-aligned overstruck combinations. As examples, **\z\(ci\(pl** will produce **⊕**, and **\(br\z\(rn\(ul\(br** will produce the smallest possible constructed box □.

12.3. Large Brackets. The Special Mathematical Font contains a number of bracket construction pieces (⎧ ⎩ ⎫ ⎭ ⎨ ⎬ │ ⌊ ⌋ ⌈ ⌉) that can be combined into various bracket styles. The function **\b'***string***'** may be used to pile up vertically the characters in *string* (the first character on top and the last at the bottom); the characters are vertically separated by 1 em and the total pile is centered 1/2 em above the current baseline (½ line in NROFF). For example, **\b'\(lc\(lf'E\|\b'\(rc\(rf'\x'−0.5m'\x'0.5m'** produces [**E**].

12.4. Line drawing. The function **\l'***Nc***'** will draw a string of repeated *c*'s towards the right for a distance *N*. (**\l** is \(lower case L). If *c* looks like a continuation of an expression for *N*, it may insulated from *N* with a **\&**. If *c* is not specified, the _ (baseline rule) is used (underline character in NROFF). If *N* is negative, a backward horizontal motion of size *N* is made *before* drawing the string. Any space resulting from *N*/(size of *c*) having a remainder is put at the beginning (left end) of the string. In the case of characters that are designed to be connected such as baseline-rule _, underrule _, and root-en ‾, the remainder space is covered by over-lapping. If *N* is *less* than the width of *c*, a single *c* is centered on a distance *N*. As an example, a macro to underscore a string can be written

```
.de us
\\$1\l'|0\(ul'
..
```

or one to draw a box around a string

```
        .de bx
        \(br\|\\$1\|\(br\l'|0\(rn'\l'|0\(ul'
        ..
```

such that

```
        .ul "underlined words"
```

and

```
        .bx "words in a box"
```

yield underlined words and words in a box.

The function \L′*Nc*′ will draw a vertical line consisting of the (optional) character *c* stacked vertically apart 1 em (1 line in NROFF), with the first two characters overlapped, if necessary, to form a continuous line. The default character is the *box rule* | (**\(br**); the other suitable character is the *bold vertical* | (**\(bv**). The line is begun without any initial motion relative to the current base line. A positive *N* specifies a line drawn downward and a negative *N* specifies a line drawn upward. After the line is drawn *no* compensating motions are made; the instantaneous baseline is at the *end* of the line.

The horizontal and vertical line drawing functions may be used in combination to produce large boxes. The zero-width *box-rule* and the ½-em wide *underrule* were *designed* to form corners when using 1-em vertical spacings. For example the macro

```
        .de eb
        .sp −1       \"compensate for next automatic base-line spacing
        .nf          \"avoid possibly overflowing word buffer
        \h'−.5n'\L'|\\nau−1'\l'\\n(.lu+1n\(ul'\L'−|\\nau+1'\l'|0u−.5n\(ul'    \"draw box
        .fi
        ..
```

will draw a box around some text whose beginning vertical place was saved in number register *a* (e. g. using **.mk a**) as done for this paragraph.

13. Hyphenation.

The automatic hyphenation may be switched off and on. When switched on with **hy**, several variants may be set. A *hyphenation indicator* character may be imbedded in a word to specify desired hyphenation points, or may be prepended to suppress hyphenation. In addition, the user may specify a small exception word list.

Only words that consist of a central alphabetic string surrounded by (usually null) non-alphabetic strings are considered candidates for automatic hyphenation. Words that were input containing hyphens (minus), em-dashes (**\(em**), or hyphenation indicator characters—such as mother-in-law—are *always* subject to splitting after those characters, whether or not automatic hyphenation is on or off.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.nh	hyphenate	-	E	Automatic hyphenation is turned off.
.hy*N*	on,*N*=1	on,*N*=1	E	Automatic hyphenation is turned on for *N*≥1, or off for *N*=0. If *N*=2, *last* lines (ones that will cause a trap) are not hyphenated. For *N*=4 and 8, the last and first two characters respectively of a word are not split off. These values are additive; i. e. *N*=14 will invoke all three restrictions.
.hc *c*	**\%**	**\%**	E	Hyphenation indicator character is set to *c* or to the default **\%**. The indicator does not appear in the output.
.hw *word1* ...		ignored	-	Specify hyphenation points in words with imbedded minus signs. Versions of a word with terminal *s* are

implied; i. e. *dig−it* implies *dig−its*. This list is examined initially *and* after each suffix stripping. The space available is small—about 128 characters.

14. Three Part Titles.

The titling function **tl** provides for automatic placement of three fields at the left, center, and right of a line with a title-length specifiable with **lt**. **tl** may be used anywhere, and is independent of the normal text collecting process. A common use is in header and footer macros.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.tl *'left'center'right'*		-	-	The strings *left*, *center*, and *right* are respectively left-adjusted, centered, and right-adjusted in the current title-length. Any of the strings may be empty, and overlapping is permitted. If the page-number character (initially %) is found within any of the fields it is replaced by the current page number having the format assigned to register %. Any character may be used as the string delimiter.
.pc *c*	%	off	-	The page number character is set to *c*, or removed. The page-number register remains **%**.
.lt ±*N*	6.5 in	previous	E,**m**	Length of title set to ±*N*. The line-length and the title-length are *independent*. Indents do not apply to titles; page-offsets do.

15. Output Line Numbering.

Automatic sequence numbering of output lines may be requested with **nm**. When in effect, a
three-digit, arabic number plus a digit-space is prepended to output text lines. The text lines are
3 thus offset by four digit-spaces, and otherwise retain their line length; a reduction in line length
may be desired to keep the right margin aligned with an earlier margin. Blank lines, other vertical
spaces, and lines generated by **tl** are *not* numbered. Numbering can be temporarily suspended with
6 **nn**, or with an **.nm** followed by a later **.nm +0**. In addition, a line number indent *I*, and the
number-text separation *S* may be specified in digit-spaces. Further, it can be specified that only
those line numbers that are multiples of some number *M* are to be printed (the others will appear
9 as blank number fields).

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.nm ±*N M S I*		off	E	Line number mode. If ±*N* is given, line numbering is turned on, and the next output line numbered is numbered ±*N*. Default values are *M*=1, *S*=1, and *I*=0. Parameters corresponding to missing arguments are unaffected; a non-numeric argument is considered missing. In the absence of all arguments, numbering is turned off; the next line number is preserved for possible further use in number register **ln**.
.nn *N*	-	*N*=1	E	The next *N* text output lines are not numbered.

As an example, the paragraph portions of this section are numbered with *M*=3: **.nm 1 3** was
placed at the beginning; **.nm** was placed at the end of the first paragraph; and **.nm +0** was placed
12 in front of this paragraph; and **.nm** finally placed at the end. Line lengths were also changed (by
\w'0000'u) to keep the right side aligned. Another example is **.nm +5 5 x 3** which turns on
numbering with the line number of the next line to be 5 greater than the last numbered line, with
15 *M*=5, with spacing *S* untouched, and with the indent *I* set to 3.

16. Conditional Acceptance of Input

In the following, *c* is a one-character, built-in *condition* name, ! signifies *not*, *N* is a numerical expression, *string1* and *string2* are strings delimited by any non-blank, non-numeric character *not* in the strings, and *anything* represents what is conditionally accepted.

Request Form	*Initial Value*	*If No Argument*	*Notes*	*Explanation*
.if *c anything*		-	-	If condition *c* true, accept *anything* as input; in multi-line case use \\{*anything*\\}.
.if !*c anything*		-	-	If condition *c* false, accept *anything.*
.if *N anything*		-	**u**	If expression $N > 0$, accept *anything.*
.if !*N anything*		-	**u**	If expression $N \leqslant 0$, accept *anything.*
.if *'string1'string2' anything*			-	If *string1* identical to *string2*, accept *anything.*
.if !*'string1'string2' anything*			-	If *string1* not identical to *string2*, accept *anything.*
.ie *c anything*		-	**u**	If portion of if-else; all above forms (like **if**).
.el *anything*		-	-	Else portion of if-else.

The built-in condition names are:

Condition Name	True If
o	Current page number is odd
e	Current page number is even
t	Formatter is TROFF
n	Formatter is NROFF

If the condition *c* is *true*, or if the number *N* is greater than zero, or if the strings compare identically (including motions and character size and font), *anything* is accepted as input. If a ! precedes the condition, number, or string comparison, the sense of the acceptance is reversed.

Any spaces between the condition and the beginning of *anything* are skipped over. The *anything* can be either a single input line (text, macro, or whatever) or a number of input lines. In the multi-line case, the first line must begin with a left delimiter \\{ and the last line must end with a right delimiter \\}.

The request **ie** (if-else) is identical to **if** except that the acceptance state is remembered. A subsequent and matching **el** (else) request then uses the reverse sense of that state. **ie** - **el** pairs may be nested.

Some examples are:

```
.if e .tl 'Even Page %'''
```

which outputs a title if the page number is even; and

```
.ie \n%>1 \{\
'sp 0.5i
.tl 'Page %'''
'sp |1.2i \}
.el .sp |2.5i
```

which treats page 1 differently from other pages.

17. Environment Switching.

A number of the parameters that control the text processing are gathered together into an *environment*, which can be switched by the user. The environment parameters are those associated with requests noting E in their *Notes* column; in addition, partially collected lines and words are in the environment. Everything else is global; examples are page-oriented parameters, diversion-oriented parameters, number registers, and macro and string definitions. All environments are initialized with default parameter values.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.ev *N*	*N*=0	previous	-	Environment switched to environment $0 \leq N \leq 2$. Switching is done in push-down fashion so that restoring a previous environment *must* be done with **.ev** rather than specific reference.

18. Insertions from the Standard Input

The input can be temporarily switched to the system *standard input* with **rd**, which will switch back when *two* newlines in a row are found (the *extra* blank line is not used). This mechanism is intended for insertions in form-letter-like documentation. On UNIX, the *standard input* can be the user's keyboard, a *pipe*, or a *file*.

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.rd *prompt*	-	*prompt*=BEL	-	Read insertion from the standard input until two newlines in a row are found. If the standard input is the user's keyboard, *prompt* (or a BEL) is written onto the user's terminal. **rd** behaves like a macro, and arguments may be placed after *prompt*.
.ex	-	-	-	Exit from NROFF/TROFF. Text processing is terminated exactly as if all input had ended.

If insertions are to be taken from the terminal keyboard *while* output is being printed on the terminal, the command line option **−q** will turn off the echoing of keyboard input and prompt only with BEL. The regular input and insertion input *cannot* simultaneously come from the standard input.

As an example, multiple copies of a form letter may be prepared by entering the insertions for all the copies in one file to be used as the standard input, and causing the file containing the letter to reinvoke itself using **nx** (§19); the process would ultimately be ended by an **ex** in the insertion file.

19. Input/Output File Switching

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.so *filename*		-	-	Switch source file. The top input (file reading) level is switched to *filename*. The effect of an **so** encountered in a macro is not felt until the input level returns to the file level. When the new file ends, input is again taken from the original file. **so**'s may be nested.
.nx *filename*		end-of-file	-	Next file is *filename*. The current file is considered ended, and the input is immediately switched to *filename*.
.pi *program*		-	-	Pipe output to *program* (NROFF only). This request must occur *before* any printing occurs. No arguments are transmitted to *program*.

20. Miscellaneous

Request Form	***Initial Value***	***If No Argument***	***Notes***	***Explanation***
.mc *c N*	-	off	E,m	Specifies that a *margin* character *c* appear a distance *N* to the right of the right margin after each non-empty text line (except those produced by **tl**). If the output line is too-long (as can happen in nofill mode) the character will

				be appended to the line. If *N* is not given, the previous *N* is used; the initial *N* is 0.2 inches in NROFF and 1 em in TROFF. The margin character used with this paragraph was a 12-point box-rule.
.tm *string*	-	newline	-	After skipping initial blanks, *string* (rest of the line) is read in *copy mode* and written on the user's terminal.
.ig *yy*	-	.*yy*=..	-	Ignore input lines. **ig** behaves exactly like **de** (§7) except that the input is discarded. The input is read in *copy mode*, and any auto-incremented registers will be affected.
.pm *t*	-	all	-	Print macros. The names and sizes of all of the defined macros and strings are printed on the user's terminal; if *t* is given, only the total of the sizes is printed. The sizes is given in *blocks* of 128 characters.
.fl	-	-	B	Flush output buffer. Used in interactive debugging to force output.

21. Output and Error Messages.

The output from **tm**, **pm**, and the prompt from **rd**, as well as various *error* messages are written onto UNIX's *standard message* output. The latter is different from the *standard output*, where NROFF formatted output goes. By default, both are written onto the user's terminal, but they can be independently redirected.

Various *error* conditions may occur during the operation of NROFF and TROFF. Certain less serious errors having only local impact do not cause processing to terminate. Two examples are *word overflow*, caused by a word that is too large to fit into the word buffer (in fill mode), and *line overflow*, caused by an output line that grew too large to fit in the line buffer; in both cases, a message is printed, the offending excess is discarded, and the affected word or line is marked at the point of truncation with a * in NROFF and a ☞ in TROFF. The philosophy is to continue processing, if possible, on the grounds that output useful for debugging may be produced. If a serious error occurs, processing terminates, and an appropriate message is printed. Examples are the inability to create, read, or write files, and the exceeding of certain internal limits that make future output unlikely to be useful.

TUTORIAL EXAMPLES

T1. Introduction

Although NROFF and TROFF have by design a syntax reminiscent of earlier text processors* with the intent of easing their use, it is almost always necessary to prepare at least a small set of macro definitions to describe most documents. Such common formatting needs as page margins and footnotes are deliberately not built into NROFF and TROFF. Instead, the macro and string definition, number register, diversion, environment switching, page-position trap, and conditional input mechanisms provide the basis for user-defined implementations.

The examples to be discussed are intended to be useful and somewhat realistic, but won't necessarily cover all relevant contingencies. Explicit numerical parameters are used in the examples to make them easier to read and to illustrate typical values. In many cases, number registers would really be used to reduce the number of places where numerical information is kept, and to concentrate conditional parameter initialization like that which depends on whether TROFF or NROFF is being used.

T2. Page Margins

As discussed in §3, *header* and *footer* macros are usually defined to describe the top and bottom page margin areas respectively. A trap is planted at page position 0 for the header, and at $-N$ (N from the page bottom) for the footer. The simplest such definitions might be

```
.de hd          \"define header
'sp 1i
..              \"end definition
.de fo          \"define footer
'bp
..              \"end definition
.wh 0 hd
.wh −1i fo
```

which provide blank 1 inch top and bottom margins. The header will occur on the *first* page, only if the definition and trap exist prior to the initial pseudo-page transition (§3). In fill mode, the output line that springs the footer trap was typically forced out because some part or whole word didn't fit on it. If anything in the footer and header that follows causes a *break*, that word or part word will be forced out. In this and other examples, requests like **bp** and **sp** that normally cause breaks are invoked using the *no-break* control character ′ to avoid this. When the header/footer design contains material requiring independent text processing, the environment may be switched, avoiding most interaction with the running text.

A more realistic example would be

```
.de hd            \"header
.if t .tl ′\(rn″\(rn′  \"troff cut mark
.if \\n%>1 \{\
′sp |0.5i−1       \"tl base at 0.5i
.tl ″− % −″       \"centered page number
.ps               \"restore size
.ft               \"restore font
.vs  \}           \"restore vs
′sp |1.0i         \"space to 1.0i
.ns               \"turn on no-space mode
..
.de fo            \"footer
.ps 10            \"set footer/header size
.ft R             \"set font
.vs 12p           \"set base-line spacing
.if \\n%=1 \{\
′sp |\\n(.pu−0.5i−1  \"tl base 0.5i up
.tl ″− % −″ \}  \"first page number
′bp
..
.wh 0 hd
.wh −1i fo
```

which sets the size, font, and base-line spacing for the header/footer material, and ultimately restores them. The material in this case is a page number at the bottom of the first page and at the top of the remaining pages. If TROFF is used, a *cut mark* is drawn in the form of *root-en*'s at each margin. The **sp**'s refer to absolute positions to avoid dependence on the base-line spacing. Another reason for this in the footer is that the footer is invoked by printing a line whose vertical spacing swept past the trap position by possibly as

*For example: P. A. Crisman, Ed., *The Compatible Time-Sharing System*, MIT Press, 1965, Section AH9.01 (Description of RUNOFF program on MIT's CTSS system).

much as the base-line spacing. The *no-space* mode is turned on at the end of **hd** to render ineffective accidental occurrences of **sp** at the top of the running text.

The above method of restoring size, font, etc. presupposes that such requests (that set *previous* value) are *not* used in the running text. A better scheme is save and restore both the current *and* previous values as shown for size in the following:

```
.de fo
.nr s1 \\n(.s    \"current size
.ps
.nr s2 \\n(.s    \"previous size
. ---            \"rest of footer
..
.de hd
. ---            \"header stuff
.ps \\n(s2       \"restore previous size
.ps \\n(s1       \"restore current size
..
```

Page numbers may be printed in the bottom margin by a separate macro triggered during the footer's page ejection:

```
.de bn           \"bottom number
.tl ''– % –''    \"centered page number
..
.wh −0.5i−1v bn \"tl base 0.5i up
```

T3. Paragraphs and Headings

The housekeeping associated with starting a new paragraph should be collected in a paragraph macro that, for example, does the desired preparagraph spacing, forces the correct font, size, base-line spacing, and indent, checks that enough space remains for *more than one* line, and requests a temporary indent.

```
.de pg           \"paragraph
.br              \"break
.ft R            \"force font,
.ps 10           \"size,
.vs 12p          \"spacing,
.in 0            \"and indent
.sp 0.4          \"prespace
.ne 1+\\n(.Vu \"want more than 1 line
.ti 0.2i         \"temp indent
..
```

The first break in **pg** will force out any previous partial lines, and must occur before the **vs**. The forcing of font, etc. is partly a defense against prior error and partly to permit things like section heading macros to set parameters only once.

The prespacing parameter is suitable for TROFF; a larger space, at least as big as the output device vertical resolution, would be more suitable in NROFF. The choice of remaining space to test for in the **ne** is the smallest amount greater than one line (the **.V** is the available vertical resolution).

A macro to automatically number section headings might look like:

```
.de sc           \"section
. ---            \"force font, etc.
.sp 0.4          \"prespace
.ne 2.4+\\n(.Vu \"want 2.4+ lines
.fi
\\n+S.
..
.nr S 0 1        \"init S
```

The usage is **.sc**, followed by the section heading text, followed by **.pg**. The **ne** test value includes one line of heading, 0.4 line in the following **pg**, and one line of the paragraph text. A word consisting of the next section number and a period is produced to begin the heading line. The format of the number may be set by **af** (§8).

Another common form is the labeled, indented paragraph, where the label protrudes left into the indent space.

```
.de lp           \"labeled paragraph
.pg
.in 0.5i         \"paragraph indent
.ta 0.2i 0.5i    \"label, paragraph
.ti 0
\t\\$1\t\c       \"flow into paragraph
..
```

The intended usage is "**.lp** *label*"; *label* will begin at 0.2 inch, and cannot exceed a length of 0.3 inch without intruding into the paragraph. The label could be right adjusted against 0.4 inch by setting the tabs instead with **.ta 0.4iR 0.5i**. The last line of **lp** ends with **\c** so that it will become a part of the first line of the text that follows.

T4. Multiple Column Output

The production of multiple column pages requires the footer macro to decide whether it was invoked by other than the last column, so that it will begin a new column rather than produce the bottom margin. The header can initialize a column register that the footer will increment and test. The following is arranged for two columns, but is easily modified for more.

```
.de hd          \"header
. ---
.nr cl 0 1      \"init column count
.mk             \"mark top of text
..
.de fo          \"footer
.ie \\n+(cl<2 \{\
.po +3.4i       \"next column; 3.1+0.3
.rt             \"back to mark
.ns \}          \"no-space mode
.el \{\
.po \\nMu       \"restore left margin
. ---
'bp \}
..
.ll 3.1i        \"column width
.nr M \\n(.o    \"save left margin
```

Typically a portion of the top of the first page contains full width text; the request for the narrower line length, as well as another **.mk** would be made where the two column output was to begin.

T5. Footnote Processing

The footnote mechanism to be described is used by imbedding the footnotes in the input text at the point of reference, demarcated by an initial **.fn** and a terminal **.ef**:

```
.fn
Footnote text and control lines...
.ef
```

In the following, footnotes are processed in a separate environment and diverted for later printing in the space immediately prior to the bottom margin. There is provision for the case where the last collected footnote doesn't completely fit in the available space.

```
.de hd          \"header
. ---
.nr x 0 1       \"init footnote count
.nr y 0-\\nb    \"current footer place
.ch fo -\\nbu   \"reset footer trap
.if \\n(dn .fz  \"leftover footnote
..
.de fo          \"footer
.nr dn 0        \"zero last diversion size
.if \\nx \{\
.ev 1           \"expand footnotes in ev1
.nf             \"retain vertical size
.FN             \"footnotes
.rm FN          \"delete it
.if "\\n(.z"fy" .di \"end overflow diversion
.nr x 0         \"disable fx
.ev \}          \"pop environment
. ---
'bp
..
.de fx          \"process footnote overflow
.if \\nx .di fy \"divert overflow
..
.de fn          \"start footnote
.da FN          \"divert (append) footnote
.ev 1           \"in environment 1
.if \\n+x=1 .fs \"if first, include separator
.fi             \"fill mode
..
.de ef          \"end footnote
.br             \"finish output
.nr z \\n(.v    \"save spacing
.ev             \"pop ev
.di             \"end diversion
.nr y -\\n(dn \"new footer position,
.if \\nx=1 .nr y -(\\n(.v-\\nz) \
                \"uncertainty correction
.ch fo \\nyu    \"y is negative
.if (\\n(nl+1v)>(\\n(.p+\\ny) \
.ch fo \\n(nlu+1v \"it didn't fit
..
.de fs          \"separator
\l'1i'          \"1 inch rule
.br
..
.de fz          \"get leftover footnote
.fn
.nf             \"retain vertical size
.fy             \"where fx put it
.ef
..
.nr b 1.0i      \"bottom margin size
.wh 0 hd        \"header trap
.wh 12i fo      \"footer trap, temp position
.wh -\\nbu fx \"fx at footer position
.ch fo -\\nbu \"conceal fx with fo
```

The header **hd** initializes a footnote count register **x**, and sets both the current footer trap position register y and the footer trap itself to a nominal position specified in register **b**. In addition, if the register **dn** indicates a leftover footnote, **fz** is invoked to reprocess it. The footnote start macro **fn** begins a diversion (append) in environment 1, and increments the count **x**; if the count is one, the footnote separator **fs** is interpolated. The separator is kept in a separate macro to permit user redefinition. The footnote end macro **ef** restores the previous environment and ends the diversion after saving the spacing size in register **z**. y is then decremented by the size of the footnote, available in **dn**; then on the first footnote, y is further decremented by the difference in vertical base-line spacings of the two environments, to prevent the late triggering the footer trap from causing the last line of the combined footnotes to overflow. The footer trap is then set to the lower (on the page) of **y** or the current page position (**nl**) plus one line, to allow for printing the reference line. If indicated by **x**, the footer **fo** rereads the footnotes from **FN** in nofill mode in environment 1, and deletes **FN**. If the footnotes were too large to fit, the macro **fx** will be trap-invoked to redivert the overflow into **fy**, and the register **dn** will later indicate to the header whether **fy** is empty. Both **fo** and **fx** are planted in the nominal footer trap position in an order that causes **fx** to be concealed unless the **fo** trap is moved. The footer then terminates the overflow diversion, if necessary, and zeros **x** to disable **fx**, because the uncertainty correction together with a not-too-late triggering of the footer can result in the footnote rereading finishing before reaching the **fx** trap.

A good exercise for the student is to combine the multiple-column and footnote mechanisms.

T6. The Last Page

After the last input file has ended, NROFF and TROFF invoke the *end macro* (§7), if any, and when it finishes, eject the remainder of the page. During the eject, any traps encountered are processed normally. At the *end* of this last page, processing terminates *unless* a partial line, word, or partial word remains. If it is desired that another page be started, the end-macro

```
.de en          \"end-macro
\c
'bp
..
.em en
```

will deposit a null partial word, and effect another last page.

Table I

Font Style Examples

The following fonts are printed in 12-point, with a vertical spacing of 14-point, and with non-alphanumeric characters separated by ¼ em space. The Special Mathematical Font was specially prepared for Bell Laboratories by Graphic Systems, Inc. of Hudson, New Hampshire. The Times Roman, Italic, and Bold are among the many standard fonts available from that company.

Times Roman

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
● □ — - _ ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢ ® ©

Times Italic

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
*! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |*
● □ — - _ ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢ ® ©

Times Bold

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
! $ % & ( ) ‘ ’ * + − . , / : ; = ? [ ] |
● □ — - _ ¼ ½ ¾ ﬁ ﬂ ﬀ ﬃ ﬄ ° † ′ ¢ ® ©

Special Mathematical Font

" ´ \ ˆ _ ` ˜ / < > { } # @ + − = ∗
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ ς τ υ φ χ ψ ω
Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
√ ‾ ≥ ≤ ≡ ∼ ≃ ≠ → ← ↑ ↓ × ÷ ± ∪ ∩ ⊂ ⊃ ⊆ ⊇ ∞ ∂
§ ∇ ¬ ∫ ∝ ∅ ∈ ‡ ☛ ☚ 🔔 | ○ ⎧ ⎩ ⎫ ⎭ ⎨ ⎬ ⎪ ⎣ ⎦ ⎡ ⎤ |

Table II

Input Naming Conventions for ´, `,and − and for Non-ASCII Special Characters

Non-ASCII characters and *minus* on the standard fonts.

Char	*Input Name*	*Character Name*	*Char*	*Input Name*	*Character Name*
’	´	close quote	ﬁ	\(fi	ﬁ
‘	`	open quote	ﬂ	\(fl	ﬂ
—	\(em	3/4 Em dash	ﬀ	\(ff	ﬀ
-	−	hyphen or	ﬃ	\(Fi	ﬃ
-	\(hy	hyphen	ﬄ	\(Fl	ﬄ
−	\−	current font minus	°	\(de	degree
●	\(bu	bullet	†	\(dg	dagger
□	\(sq	square	′	\(fm	foot mark
_	\(ru	rule	¢	\(ct	cent sign
¼	\(14	1/4	®	\(rg	registered
½	\(12	1/2	©	\(co	copyright
¾	\(34	3/4			

Non-ASCII characters and ´, `, _, +, −, =, and * on the special font.

The ASCII characters @, #, ", ´, `, <, >, \, {, }, ˜, ˆ, and _ exist *only* on the special font and are printed as a 1-em space if that font is not mounted. The following characters exist only on the special font except for the upper case Greek letter names followed by † which are mapped into upper case English letters in whatever font is mounted on font position one (default Times Roman). The special math plus, minus, and equals are provided to insulate the appearance of equations from the choice of standard fonts.

Char	*Input Name*	*Character Name*	*Char*	*Input Name*	*Character Name*
+	\(pl	math plus	κ	\(*k	kappa
−	\(mi	math minus	λ	\(*l	lambda
=	\(eq	math equals	μ	\(*m	mu
*	\(**	math star	ν	\(*n	nu
§	\(sc	section	ξ	\(*c	xi
´	\(aa	acute accent	ο	\(*o	omicron
`	\(ga	grave accent	π	\(*p	pi
_	\(ul	underrule	ρ	\(*r	rho
/	\(sl	slash (matching backslash)	σ	\(*s	sigma
α	\(*a	alpha	ς	\(ts	terminal sigma
β	\(*b	beta	τ	\(*t	tau
γ	\(*g	gamma	υ	\(*u	upsilon
δ	\(*d	delta	φ	\(*f	phi
ε	\(*e	epsilon	χ	\(*x	chi
ζ	\(*z	zeta	ψ	\(*q	psi
η	\(*y	eta	ω	\(*w	omega
θ	\(*h	theta	A	\(*A	Alpha†
ι	\(*i	iota	B	\(*B	Beta†

	Input	*Character*
Char	***Name***	***Name***
Γ	\(*G	Gamma
Δ	\(*D	Delta
E	\(*E	Epsilon†
Z	\(*Z	Zeta†
H	\(*Y	Eta†
Θ	\(*H	Theta
I	\(*I	Iota†
K	\(*K	Kappa†
Λ	\(*L	Lambda
M	\(*M	Mu†
N	\(*N	Nu†
Ξ	\(*C	Xi
O	\(*O	Omicron†
Π	\(*P	Pi
P	\(*R	Rho†
Σ	\(*S	Sigma
T	\(*T	Tau†
Υ	\(*U	Upsilon
Φ	\(*F	Phi
X	\(*X	Chi†
Ψ	\(*Q	Psi
Ω	\(*W	Omega
√	\(sr	square root
‾	\(rn	root en extender
≥	\(>=	>=
≤	\(<=	<=
≡	\(==	identically equal
≃	\(~=	approx =
~	\(ap	approximates
≠	\(!=	not equal
→	\(->	right arrow
←	\(<-	left arrow
↑	\(ua	up arrow
↓	\(da	down arrow
×	\(mu	multiply
÷	\(di	divide
±	\(+-	plus-minus
∪	\(cu	cup (union)
∩	\(ca	cap (intersection)
⊂	\(sb	subset of
⊃	\(sp	superset of
⊆	\(ib	improper subset
⊇	\(ip	improper superset
∞	\(if	infinity
∂	\(pd	partial derivative
∇	\(gr	gradient
¬	\(no	not
∫	\(is	integral sign
∝	\(pt	proportional to
∅	\(es	empty set
∈	\(mo	member of

	Input	*Character*
Char	***Name***	***Name***
│	\(br	box vertical rule
‡	\(dd	double dagger
☛	\(rh	right hand
☚	\(lh	left hand
Ⓐ	\(bs	Bell System logo
│	\(or	or
○	\(ci	circle
⎧	\(lt	left top of big curly bracket
⎩	\(lb	left bottom
⎫	\(rt	right top
⎭	\(rb	right bot
⎨	\(lk	left center of big curly bracket
⎬	\(rk	right center of big curly bracket
┃	\(bv	bold vertical
⌊	\(lf	left floor (left bottom of big square bracket)
⌋	\(rf	right floor (right bottom)
⌈	\(lc	left ceiling (left top)
⌉	\(rc	right ceiling (right top)

Summary of Changes to N/TROFF Since October 1976 Manual

Options

-h	(Nroff only) Output tabs used during horizontal spacing to speed output as well as reduce output byte count. Device tab settings assumed to be every 8 nominal character widths. The default settings of input (logical) tabs is also initialized to every 8 nominal character widths.
-z	Efficiently suppresses formatted output. Only message output will occur (from "tm"s and diagnostics).

Old Requests

.ad c	The adjustment type indicator "c" may now also be a number previously obtained from the ".j" register (see below).
.so name	The contents of file "name" will be interpolated at the point the "so" is encountered. Previously, the interpolation was done upon return to the file-reading input level.

New Request

.ab text — Prints "text" on the message output and terminates without further processing. If "text" is missing, "User Abort." is printed. Does not cause a break. The output buffer is flushed.

.fz F N — forces font "F" to be in size N. N may have the form N, +N, or -N. For example,

```
.fz 3 -2
```

will cause an implicit \s-2 every time font 3 is entered, and a corresponding \s+2 when it is left. Special font characters occurring during the reign of font F will have the same size modification. If special characters are to be treated differently,

```
.fz S F N
```

may be used to specify the size treatment of special characters during font F. For example,

```
.fz 3 -3
.fz S 3 -0
```

will cause automatic reduction of font 3 by 3 points while the special characters would not be affected. Any ".fp" request specifying a font on some position must precede ".fz" requests relating to that position.

New Predefined Number Registers.

.k	Read-only. Contains the horizontal size of the text portion (without indent) of the current partially collected output line, if any, in the current environment.
.j	Read-only. A number representing the current adjustment mode and type. Can be saved and later given to the "ad" request to restore a previous mode.
.P	Read-only. 1 if the current page is being printed, and zero otherwise.
.L	Read-only. Contains the current line-spacing parameter ("ls").
c.	General register access to the input line-number in the current input file. Contains the same value as the read-only ".c" register.

Addendum: Tektronix Modifications to *Nroff/Troff*

Rick LeFaivre

Computer Research
Applied Research Group
Tektronix Labs

Introduction

This short addendum to the *NROFF/TROFF User's Manual* documents extensions made to *nroff/troff* at Tektronix during 1979-80. In addition to the modifications documented here, a number of internal improvements and bug fixes have been made to *nroff/troff*—these are all noted by TEK comments in the code.

Fonts

The "font" mechanism in *nroff* was extended to provide both boldface and a continuous underline font (which, unlike the old **.cu**, may extend over line breaks). Boldface is produced by overprinting, with a slight offset on output devices capable of fine horizontal movement. The number of overprints is given by the new number register *bn*, and the offset in minimal horizontal units is given by register *bo*. The available font numbers and names in *nroff* are now:

1	R	Roman
2	U	Underline Alphanumeric
3	B	Boldface
4	BU	Underlined Boldface
6	C	Continuous Underlined
8	BC	Continuous Underlined Boldface

An artificially emboldened pseudofont capability was also added to *troff*, with *bo* again controlling the offset amount (*bn* is ignored in *troff*). If *bo* is greater than 1 it gives the offset amount minus 1 in basic units (as in the **.bd** command). If *bo* is equal to 1, the offset amount is computed as \n(.s/2 units, which gives a pleasing offset for a variety of point sizes. This convention is now also used if 1 is given as the offset amount in the **.bd** command. The font numbers and names in *troff* are now:

1	R	Roman
2	I	Italic
3	B	Boldface
4	S	Special
5	-	Artificial Bold Roman
6	BI	Artificial Bold Italics
7	-	Artificial Bold Boldface

Note that because of differences between *nroff* and *troff* it is desirable to use font names instead of absolute numbers, especially for font numbers 5-8. *troff* font I is automatically mapped to font U in *nroff*, and *nroff* fonts U and C are automatically mapped to font I in *troff*.

The command **.fp** was modified slightly to allow font pseudonames to be given without reading a new font width table—simply add an extra X to the end, as in:

```
.fp 5 BR X
```

This allows font 5 (fake bold) to be referenced via \f(BR. If the X weren't given, a width table for new font BR would have to be provided.

Terminal Driving Tables

The *nroff* terminal driving tables were modified to include a *termtype* entry which gives the type of terminal being used. The terminal type is accessed via the .T register; currently assigned types are:

0	Standard ASCII (Default)
1	Printronix Line Printer
2	Vanilla Line Printer
3	Qume SPRINT-5
4	Tek 4025
5	DEC VT-100 / Plessey PT-100
6	Diablo HyType 450

A macro package may thus tailor itself to a particular terminal via a sequence like:

```
.if\n(.T==3 . . .
```

All other terminal types currently have a *termtype* of 100 (to be interpreted as "no special support at this site". These may be changed as desired. Entries for automatic underline attribute codes (*ulon, uloff*) were also added to complement the existing *bdon* and *bdoff.* If these sequences are used they are now inserted before every underlined or emboldened character so that terminals which disable selected attributes at the end of a line (like the Tek 4025) work properly. Several modifications were also made to support the Qume SPRINT-5, including the use of absolute positioning commands for interword spacing.

To support these changes to the terminal driving tables while allowing the original Bell *nroff* to run, the new tables are stored in */usr/lib/nterm* rather than */usr/lib/term.*

New Commands

.dn *name1 name2*

Defines *name1* to be a pseudoname for command or macro *name2.* For example,

```
.dn SP sp
```

.sy *system call*

Passes *system call* on to the UNIX shell to be executed, then returns to *nroff/troff.*

.df *file*
.dx *file*

.df is like .di, except the output is directed to a file. .dx is the append version. As a simple example, a list of words may be sorted via something like:

```
.nf
.df /tmp/sort\n(id
word1
word2
. . .
.df
.sy sort /tmp/sort\n(id -o /tmp/sort\n(id
.so /tmp/sort\n(id
.sy rm -f /tmp/sort\n(id
```

These commands are used in the Tektronix *ms* package to support the creation of a sorted index. (The *ms* index and table of contents also required several modifications to allow **.so** and other commands to be invoked from the **.em** end macro.)

New In-Line Commands

\\$*	Inside a macro, interpolate *all* arguments on the command line, separated by spaces. Note that there may now be more than nine arguments—\\$* interpolates them all, and \\$9 interpolates the 9th and all succeeding arguments (again, separated by spaces).
\i*n'string'*	Interpolates the *n*th character in *string*. For example, to check if the first character of a macro argument is '+', one could do: `.if "\i1'\\$1'"+" . . .` If *string* is less than *n* characters long, the null string is returned.
\F*font*	Same as \f*font*.

New Number Registers

\n(id	The current process id. Useful for creating temporary file names for .df.
\n(bn	Number of bold overprints—initially 6.
\n(bo	Bold offset—initially 1.
\n(.T	In *nroff*, gives the terminal type (0 in *troff*).
\n(.e	1 if a forced page eject (.bp) is being done. Allows trap macros to decide if a real page eject was requested, or if text flowed into the trap. Used, for example, to allow .bp to work properly even if one is in the first column of a two-column page.

Section 5

USING THE -MS MACROS WITH TROFF AND NROFF

INTRODUCTION

The *-ms* macro package was developed at Bell Laboratories and is licensed by Western Electric for use on the 8560. The remainder of this section contains a reprint of an article describing the *-ms* macro package and an addendum detailing the version supplied by Tektronix. The Technical Notes section of this manual describes the limitations of this package and any changes made to this package by Tektronix.

Typing Documents on the UNIX System: Using the −ms Macros with Troff and Nroff

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This document describes a set of easy-to-use macros for preparing documents on the UNIX system. Documents may be produced on either the phototypesetter or a on a computer terminal, without changing the input.

The macros provide facilities for paragraphs, sections (optionally with automatic numbering), page titles, footnotes, equations, tables, two-column format, and cover pages for papers.

This memo includes, as an appendix, the text of the "Guide to Preparing Documents with −ms" which contains additional examples of features of −ms.

This manual is a revision of, and replaces, "Typing Documents on UNIX," dated November 22, 1974.

November 13, 1978

Typing Documents on the UNIX System: Using the −ms Macros with Troff and Nroff

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

Introduction. This memorandum describes a package of commands to produce papers using the *troff* and *nroff* formatting programs on the UNIX system. As with other *roff*-derived programs, text is prepared interspersed with formatting commands. However, this package, which itself is written in *troff* commands, provides higher-level commands than those provided with the basic *troff* program. The commands available in this package are listed in Appendix A.

Text. Type normally, except that instead of indenting for paragraphs, place a line reading ".PP" before each paragraph. This will produce indenting and extra space.

Alternatively, the command .LP that was used here will produce a left-aligned (block) paragraph. The paragraph spacing can be changed: see below under "Registers."

Beginning. For a document with a paper-type cover sheet, the input should start as follows:

```
[optional overall format .RP — see below]
.TL
Title of document (one or more lines)
.AU
Author(s) (may also be several lines)
.AI
Author's institution(s)
.AB
Abstract; to be placed on the cover sheet of a paper.
Line length is 5/6 of normal; use .ll here to change.
.AE (abstract end)
text ... (begins with .PP, which see)
```

To omit some of the standard headings (e.g. no abstract, or no author's institution) just omit the corresponding fields and command lines. The word ABSTRACT can be suppressed by writing ".AB no" for ".AB". Several interspersed .AU and .AI lines can be used for multiple authors. The headings are not compulsory: beginning with a .PP command is perfectly OK and will just start printing an ordinary paragraph. ***Warning:*** You can't just begin a document with a line of text. Some −ms command must precede any text input. When in doubt, use .LP to get proper initialization, although any of the commands .PP, .LP, .TL, .SH, .NH is good enough. Figure 1 shows the legal arrangement of commands at the start of a document.

Cover Sheets and First Pages. The first line of a document signals the general format of the first page. In particular, if it is ".RP" a cover sheet with title and abstract is prepared. The default format is useful for scanning drafts.

In general −ms is arranged so that only one form of a document need be stored, containing all information; the first command gives the format, and unnecessary items for that format are ignored.

Warning: don't put extraneous material between the .TL and .AE commands. Processing of the titling items is special, and other data placed in them may not behave as you expect. Don't forget that some −ms command must precede any input text.

Page headings. The −ms macros, by default, will print a page heading containing a page number (if greater than 1). A default page footer is provided only in ***nroff***, where the date is used. The user can make minor adjustments to the page headings/footings by redefining the strings LH, CH, and RH which are the left, center and right portions of the page headings, respectively; and the strings LF, CF, and RF, which are the left, center and right portions of the page footer. For more complex formats, the user can redefine the macros PT and BT, which are invoked respectively at the top and bottom of each page. The margins (taken from registers HM and FM for the top and bottom margin respectively) are normally 1 inch; the page header/footer are in the middle of that space. The user who redefines these macros should be careful not to change parameters such as point size or font without resetting them to default values.

Multi-column formats. If you place the command ".2C" in your document, the document will be printed in double column format beginning at that point. This feature is not too useful in computer terminal output, but is often desirable on the typesetter. The command ".1C" will go back to one-column format and also skip to a new page. The ".2C" command is actually a special case of the command

```
.MC [column width [gutter width]]
```

which makes multiple columns with the specified column and gutter width; as many columns as will fit across the page are used. Thus triple, quadruple, ... column pages can be printed. Whenever the number of columns is changed (except going from full width to some larger number of columns) a new page is started.

Headings. To produce a special heading, there are two commands. If you type

```
.NH
type section heading here
may be several lines
```

you will get automatically numbered section headings (1, 2, 3, ...), in boldface. For example,

```
.NH
Care and Feeding of Department Heads
```

produces

1. Care and Feeding of Department Heads

Alternatively,

```
.SH
Care and Feeding of Directors
```

will print the heading with no number added:

Care and Feeding of Directors

Every section heading, of either type, should be followed by a paragraph beginning with .PP or .LP, indicating the end of the heading. Headings may contain more than one line of text.

The .NH command also supports more complex numbering schemes. If a numerical argument is given, it is taken to be a "level" number and an appropriate sub-section number is generated. Larger level numbers indicate deeper sub-sections, as in this example:

```
.NH
Erie-Lackawanna
.NH 2
Morris and Essex Division
.NH 3
Gladstone Branch
.NH 3
Montclair Branch
.NH 2
Boonton Line
```

generates:

2. Erie-Lackawanna

2.1. Morris and Essex Division

2.1.1. Gladstone Branch

2.1.2. Montclair Branch

2.2. Boonton Line

An explicit ".NH 0" will reset the numbering of level 1 to one, as here:

```
.NH 0
Penn Central
```

1. Penn Central

Indented paragraphs. (Paragraphs with hanging numbers, e.g. references.) The sequence

```
.IP [1]
Text for first paragraph, typed
normally for as long as you would
like on as many lines as needed.
.IP [2]
Text for second paragraph, ...
```

produces

[1] Text for first paragraph, typed normally for as long as you would like on as many lines as needed.

[2] Text for second paragraph, ...

A series of indented paragraphs may be followed by an ordinary paragraph beginning with .PP or .LP, depending on whether you wish indenting or not. The command .LP was used here.

More sophisticated uses of .IP are also possible. If the label is omitted, for example, a plain block indent is produced.

```
.IP
This material will
just be turned into a
block indent suitable for quotations or
such matter.
.LP
```

will produce

> This material will just be turned into a block indent suitable for quotations or such matter.

If a non-standard amount of indenting is required, it may be specified after the label (in character positions) and will remain in effect until the next .PP or .LP. Thus, the general form of the .IP command contains two additional fields: the label and the indenting length. For example,

```
.IP first: 9
Notice the longer label, requiring larger
indenting for these paragraphs.
.IP second:
And so forth.
.LP
```

produces this:

first: Notice the longer label, requiring larger indenting for these paragraphs.

second: And so forth.

It is also possible to produce multiple nested indents; the command .RS indicates that the next .IP starts from the current indentation level. Each .RE will eat up one level of indenting so you should balance .RS and .RE commands. The .RS command should be thought of as "move right" and the .RE command as "move left". As an example

```
.IP 1.
Bell Laboratories
.RS
.IP 1.1
Murray Hill
.IP 1.2
Holmdel
.IP 1.3
Whippany
.RS
.IP 1.3.1
Madison
.RE
.IP 1.4
Chester
.RE
.LP
```

will result in

```
1.   Bell Laboratories
     1.1  Murray Hill
     1.2  Holmdel
     1.3  Whippany
          1.3.1 Madison
     1.4  Chester
```

All of these variations on .LP leave the right margin untouched. Sometimes, for purposes such as setting off a quotation, a paragraph indented on both right and left is required.

> A single paragraph like this is obtained by preceding it with .QP. More complicated material (several paragraphs) should be bracketed with .QS and .QE.

Emphasis. To get italics (on the typesetter) or underlining (on the terminal) say

```
.I
as much text as you want
can be typed here
.R
```

as was done for *these three words.* The .R command restores the normal (usually Roman) font. If only one word is to be italicized, it may be just given on the line with the .I command,

```
.I word
```

and in this case no .R is needed to restore the previous font. **Boldface** can be produced by

```
.B
Text to be set in boldface
goes here
.R
```

and also will be underlined on the terminal or line printer. As with .I, a single word can be placed in boldface by placing it on the same line as the .B command.

A few size changes can be specified similarly with the commands .LG (make larger), .SM (make smaller), and .NL (return to normal size). The size change is two points; the commands may be repeated for increased effect (here one .NL canceled two .SM commands).

If actual underlining as opposed to italicizing is required on the typesetter, the command

```
.UL word
```

will underline a word. There is no way to underline multiple words on the typesetter.

Footnotes. Material placed between lines with the commands .FS (footnote) and .FE (footnote end) will be collected, remembered, and finally placed at the bottom of the current page*. By default, footnotes are 11/12th the length of normal text, but this can be changed using the FL register (see below).

Displays and Tables. To prepare displays of lines, such as tables, in which the lines should not be re-arranged, enclose them in the commands .DS and .DE

* Like this.

```
.DS
table lines, like the
examples here, are placed
between .DS and .DE
.DE
```

By default, lines between .DS and .DE are indented and left-adjusted. You can also center lines, or retain the left margin. Lines bracketed by .DS C and .DE commands are centered (and not re-arranged); lines bracketed by .DS L and .DE are left-adjusted, not indented, and not re-arranged. A plain .DS is equivalent to .DS I, which indents and left-adjusts. Thus,

```
these lines were preceded
  by .DS C and followed by
     a .DE command;
```

whereas

```
these lines were preceded
by .DS L and followed by
a .DE command.
```

Note that .DS C centers each line; there is a variant .DS B that makes the display into a left-adjusted block of text, and then centers that entire block. Normally a display is kept together, on one page. If you wish to have a long display which may be split across page boundaries, use .CD, .LD, or .ID in place of the commands .DS C, .DS L, or .DS I respectively. An extra argument to the .DS I or .DS command is taken as an amount to indent. Note: it is tempting to assume that .DS R will right adjust lines, but it doesn't work.

Boxing words or lines. To draw rectangular boxes around words the command

```
.BX word
```

will print $\boxed{\text{word}}$ as shown. The boxes will not be neat on a terminal, and this should not be used as a substitute for italics.

Longer pieces of text may be boxed by enclosing them with .B1 and .B2:

```
.B1
text...
.B2
```

as has been done here.

Keeping blocks together. If you wish to keep a table or other block of lines together on a page, there are "keep -

release" commands. If a block of lines preceded by .KS and followed by .KE does not fit on the remainder of the current page, it will begin on a new page. Lines bracketed by .DS and .DE commands are automatically kept together this way. There is also a "keep floating" command: if the block to be kept together is preceded by .KF instead of .KS and does not fit on the current page, it will be moved down through the text until the top of the next page. Thus, no large blank space will be introduced in the document.

Nroff/Troff commands. Among the useful commands from the basic formatting programs are the following. They all work with both typesetter and computer terminal output:

.bp - begin new page.
.br - "break", stop running text
  from line to line.
.sp n - insert n blank lines.
.na - don't adjust right margins.

Date. By default, documents produced on computer terminals have the date at the bottom of each page; documents produced on the typesetter don't. To force the date, say ".DA". To force no date, say ".ND". To lie about the date, say ".DA July 4, 1776" which puts the specified date at the bottom of each page. The command

.ND May 8, 1945

in ".RP" format places the specified date on the cover sheet and nowhere else. Place this line before the title.

Signature line. You can obtain a signature line by placing the command .SG in the document. The authors' names will be output in place of the .SG line. An argument to .SG is used as a typing identification line, and placed after the signatures. The .SG command is ignored in released paper format.

Registers. Certain of the registers used by −ms can be altered to change default settings. They should be changed with .nr commands, as with

.nr PS 9

to make the default point size 9 point. If the effect is needed immediately, the normal *troff* command should be used in addition to changing the number register.

Register	Defines	Takes effect	Default
PS	point size	next para.	10
VS	line spacing	next para.	12 pts
LL	line length	next para.	6"
LT	title length	next para.	6"
PD	para. spacing	next para.	0.3 VS
PI	para. indent	next para.	5 ens
FL	footnote length	next FS	11/12 LL
CW	column width	next 2C	7/15 LL
GW	intercolumn gap	next 2C	1/15 LL
PO	page offset	next page	26/27"
HM	top margin	next page	1"
FM	bottom margin	next page	1"

You may also alter the strings LH, CH, and RH which are the left, center, and right headings respectively; and similarly LF, CF, and RF which are strings in the page footer. The page number on *output* is taken from register PN, to permit changing its output style. For more complicated headers and footers the macros PT and BT can be redefined, as explained earlier.

Accents. To simplify typing certain foreign words, strings representing common accent marks are defined. They precede the letter over which the mark is to appear. Here are the strings:

Input	Output	Input	Output
*'e	é	*~a	ã
*\`e	è	*Ce	ě
*:u	ü	*,c	c
*^e	ê		

Use. After your document is prepared and stored on a file, you can print it on a terminal with the command*

nroff −ms file

and you can print it on the typesetter with the command

troff −ms file

(many options are possible). In each case, if your document is stored in several files, just list all the filenames where we have used "file". If equations or tables are used, *eqn* and/or *tbl* must be invoked as preprocessors.

* If .2C was used, pipe the *nroff* output through *col;* make the first line of the input ".pi /usr/bin/col."

References and further study. If you have to do Greek or mathematics, see ***eqn*** [1] for equation setting. To aid ***eqn*** users, $-$***ms*** provides definitions of .EQ and .EN which normally center the equation and set it off slightly. An argument on .EQ is taken to be an equation number and placed in the right margin near the equation. In addition, there are three special arguments to EQ: the letters C, I, and L indicate centered (default), indented, and left adjusted equations, respectively. If there is both a format argument and an equation number, give the format argument first, as in

.EQ L (1.3a)

for a left-adjusted equation numbered (1.3a).

Similarly, the macros .TS and .TE are defined to separate tables (see [2]) from text with a little space. A very long table with a heading may be broken across pages by beginning it with .TS H instead of .TS, and placing the line .TH in the table data after the heading. If the table has no heading repeated from page to page, just use the ordinary .TS and .TE macros.

To learn more about *troff* see [3] for a general introduction, and [4] for the full details (experts only). Information on related UNIX commands is in [5]. For jobs that do not seem well-adapted to $-$ms, consider other macro packages. It is often far easier to write a specific macro packages for such tasks as imitating particular journals than to try to adapt $-$ms.

Acknowledgment. Many thanks are due to Brian Kernighan for his help in the design and implementation of this package, and for his assistance in preparing this manual.

References

[1] B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics — Users Guide (2nd edition),* Bell Laboratories Computing Science Report no. 17.

[2] M. E. Lesk, *Tbl — A Program to Format Tables,* Bell Laboratories Computing Science Report no. 45.

[3] B. W. Kernighan, *A Troff Tutorial,* Bell Laboratories, 1976.

[4] J. F. Ossanna, *Nroff/Troff Reference Manual,* Bell Laboratories Computing Science Report no. 51.

[5] K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual,* Bell Laboratories, 1978.

Appendix A
List of Commands

1C	Return to single column format.
2C	Start double column format.
AB	Begin abstract.
AE	End abstract.
AI	Specify author's institution.
AU	Specify author.
B	Begin boldface.
DA	Provide the date on each page.
DE	End display.
DS	Start display (also CD, LD, ID).
EN	End equation.
EQ	Begin equation.
FE	End footnote.
FS	Begin footnote.
I	Begin italics.
IP	Begin indented paragraph.
KE	Release keep.
KF	Begin floating keep.
KS	Start keep.
LG	Increase type size.
LP	Left aligned block paragraph.
ND	Change or cancel date.
NH	Specify numbered heading.
NL	Return to normal type size.
PP	Begin paragraph.
R	Return to regular font (usually Roman).
RE	End one level of relative indenting.
RP	Use released paper format.
RS	Relative indent increased one level.
SG	Insert signature line.
SH	Specify section heading.
SM	Change to smaller type size.
TL	Specify title.
UL	Underline one word.

Register Names

The following register names are used by −ms internally. Independent use of these names in one's own macros may produce incorrect output. Note that no lower case letters are used in any −ms internal name.

Number registers used in −ms

:	DW	GW	HM	IQ	LL	NA	OJ	PO	T.	TV
#T	EF	H1	HT	IR	LT	NC	PD	PQ	TB	VS
1T	FL	H3	IK	KI	MM	NF	PF	PX	TD	YE
AV	FM	H4	IM	L1	MN	NS	PI	RO	TN	YY
CW	FP	H5	IP	LE	MO	OI	PN	ST	TQ	ZN

String registers used in −ms

'	A5	CB	DW	EZ	I	KF	MR	R1	RT	TL
`	AB	CC	DY	FA	I1	KQ	ND	R2	S0	TM
^	AE	CD	E1	FE	I2	KS	NH	R3	S1	TQ
~	AI	CF	E2	FJ	I3	LB	NL	R4	S2	TS
:	AU	CH	E3	FK	I4	LD	NP	R5	SG	TT
,	B	CM	E4	FN	I5	LG	OD	RC	SH	UL
1C	BG	CS	E5	FO	ID	LP	OK	RE	SM	WB
2C	BT	CT	EE	FQ	IE	ME	PP	RF	SN	WH
A1	C	D	EL	FS	IM	MF	PT	RH	SY	WT
A2	C1	DA	EM	FV	IP	MH	PY	RP	TA	XD
A3	C2	DE	EN	FY	IZ	MN	QF	RQ	TE	XF
A4	CA	DS	EQ	HO	KE	MO	R	RS	TH	XK

Order of Commands in Input

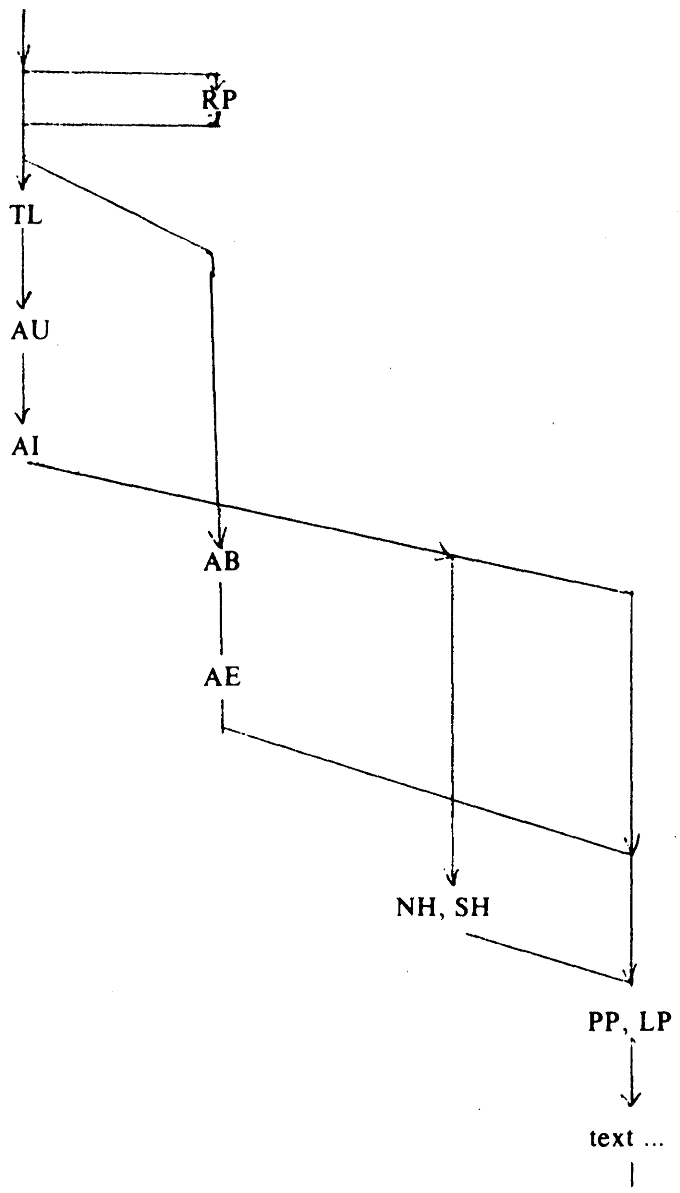


Figure 1

A Guide to Preparing Documents with −ms

M. E. Lesk

Bell Laboratories August 1978

This guide gives some simple examples of document preparation on Bell Labs computers, emphasizing the use of the *−ms* macro package. It enormously abbreviates information in

1. *Typing Documents on UNIX and GCOS,* by M. E. Lesk;
2. *Typesetting Mathematics — User's Guide,* by B. W. Kernighan and L. L. Cherry; and
3. *Tbl — A Program to Format Tables,* by M. E. Lesk.

These memos are all included in the *UNIX Programmer's Manual, Volume 2.* The new user should also have *A Tutorial Introduction to the UNIX Text Editor,* by B. W. Kernighan.

For more detailed information, read *Advanced Editing on UNIX* and *A Troff Tutorial,* by B. W. Kernighan, and (for experts) *Nroff/Troff Reference Manual* by J. F. Ossanna. Information on related commands is found (for UNIX users) in *UNIX for Beginners* by B. W. Kernighan and the *UNIX Programmer's Manual* by K. Thompson and D. M. Ritchie.

Contents

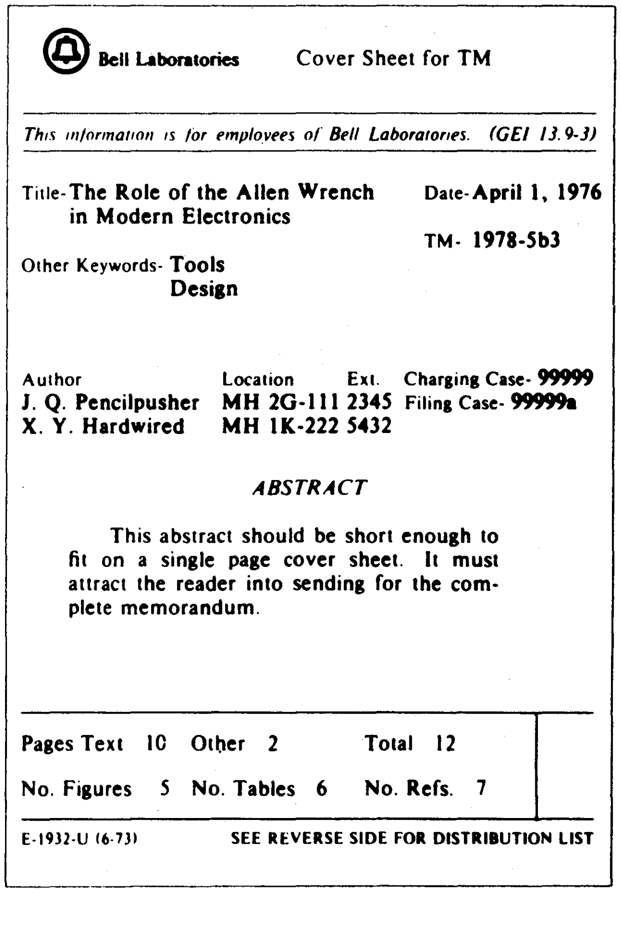
Throughout the examples, input is shown in
 this Helvetica sans serif font
while the resulting output is shown in
 this Times Roman font.

UNIX Document no. 1111

Commands for a TM

```
.TM 1978-5b3 99999 99999-11
.ND April 1, 1976
.TL
The Role of the Allen Wrench in Modern
Electronics
.AU "MH 2G-111" 2345
J. Q. Pencilpusher
.AU "MH 1K-222" 5432
X. Y. Hardwired
.AI
.MH
.OK
Tools
Design
.AB
This abstract should be short enough to
fit on a single page cover sheet.
It must attract the reader into sending for
the complete memorandum.
.AE
.CS 10 2 12 5 6 7
.NH
Introduction.
.PP
Now the first paragraph of actual text ...
...
Last line of text.
.SG MH-1234-JQP/XYH-unix
.NH
References ...
```

Commands not needed in a particular format are ignored.



Bell Laboratories Cover Sheet for TM

This information is for employees of Bell Laboratories. (GEI 13.9-3)

Title- **The Role of the Allen Wrench in Modern Electronics** Date- **April 1, 1976**

TM- **1978-5b3**

Other Keywords- **Tools Design**

Author	Location	Ext.	
J. Q. Pencilpusher	**MH 2G-111**	**2345**	Charging Case- **99999**
X. Y. Hardwired	**MH 1K-222**	**5432**	Filing Case- **99999a**

ABSTRACT

This abstract should be short enough to fit on a single page cover sheet. It must attract the reader into sending for the complete memorandum.

Pages Text 10	Other 2	Total 12
No. Figures 5	No. Tables 6	No. Refs. 7

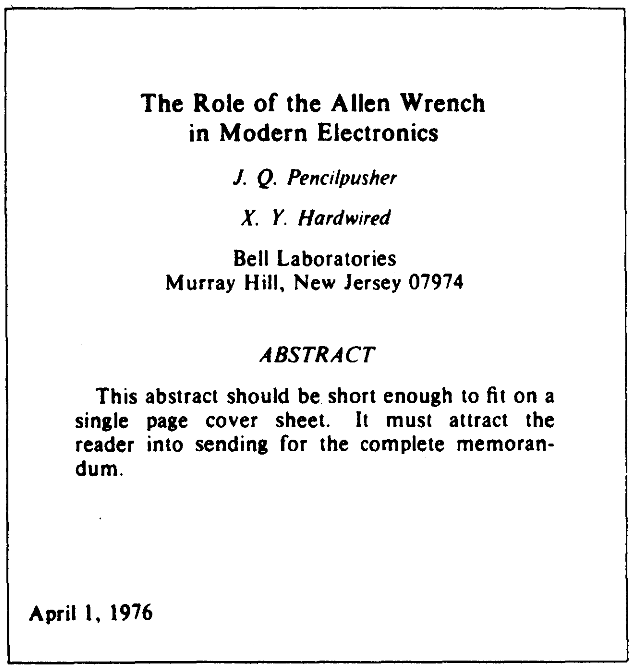
E-1932-U (6-73) **SEE REVERSE SIDE FOR DISTRIBUTION LIST**

A Released Paper with Mathematics

```
.EQ
delim $$
.EN
.RP

... (as for a TM)

.CS 10 2 12 5 6 7
.NH
Introduction
.PP
The solution to the torque handle equation
.EQ (1)
sum from 0 to inf F ( x sub i ) = G ( x )
.EN
is found with the transformation $ x = rho over
theta $ where $ rho = G prime (x) $ and $theta$
is derived ...
```



The Role of the Allen Wrench
in Modern Electronics

J. Q. Pencilpusher

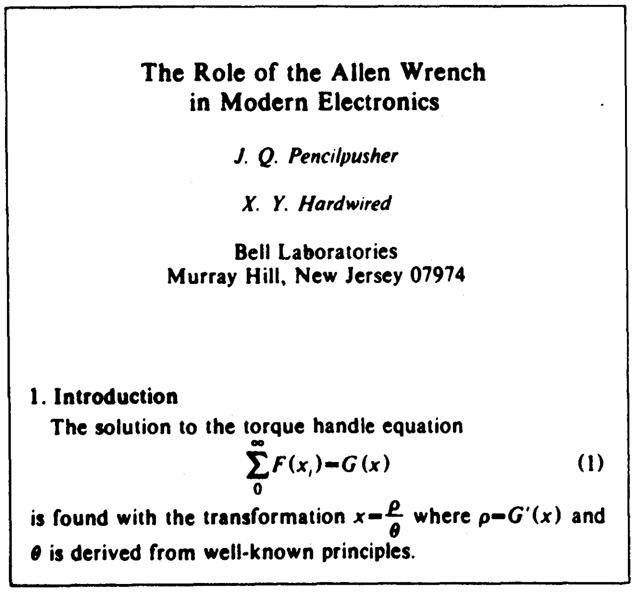
X. Y. Hardwired

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This abstract should be short enough to fit on a single page cover sheet. It must attract the reader into sending for the complete memorandum.

April 1, 1976



The Role of the Allen Wrench
in Modern Electronics

J. Q. Pencilpusher

X. Y. Hardwired

Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

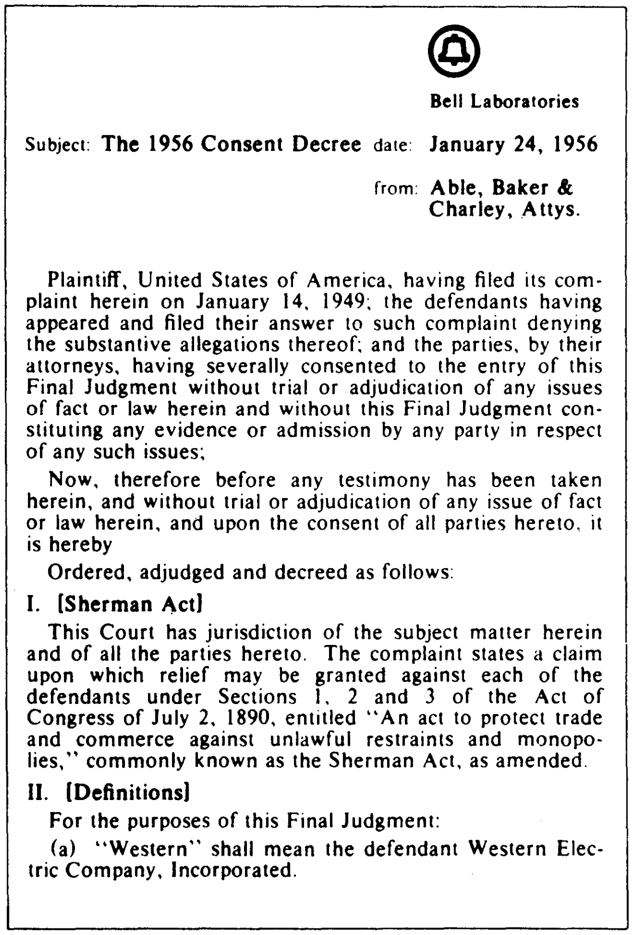
The solution to the torque handle equation

$$\sum_{0}^{\infty} F(x_i) = G(x) \qquad (1)$$

is found with the transformation $x = \frac{\rho}{\theta}$ where $\rho = G'(x)$ and θ is derived from well-known principles.

An Internal Memorandum

```
.IM
.ND January 24, 1956
.TL
The 1956 Consent Decree
.AU
Able, Baker &
Charley, Attys.
.PP
Plaintiff, United States of America, having filed
its complaint herein on January 14, 1949; the
defendants having appeared and filed their
answer to such complaint denying the
substantive allegations thereof; and the parties,
by their attorneys, ...
```



Bell Laboratories

Subject: **The 1956 Consent Decree** date: **January 24, 1956**

from: **Able, Baker &**
Charley, Attys.

Plaintiff, United States of America, having filed its complaint herein on January 14, 1949; the defendants having appeared and filed their answer to such complaint denying the substantive allegations thereof; and the parties, by their attorneys, having severally consented to the entry of this Final Judgment without trial or adjudication of any issues of fact or law herein and without this Final Judgment constituting any evidence or admission by any party in respect of any such issues;

Now, therefore before any testimony has been taken herein, and without trial or adjudication of any issue of fact or law herein, and upon the consent of all parties hereto, it is hereby

Ordered, adjudged and decreed as follows:

I. [Sherman Act]

This Court has jurisdiction of the subject matter herein and of all the parties hereto. The complaint states a claim upon which relief may be granted against each of the defendants under Sections 1, 2 and 3 of the Act of Congress of July 2, 1890, entitled "An act to protect trade and commerce against unlawful restraints and monopolies," commonly known as the Sherman Act, as amended.

II. [Definitions]

For the purposes of this Final Judgment:

(a) "Western" shall mean the defendant Western Electric Company, Incorporated.

Other formats possible (specify before .TL) are: .MR ("memo for record"), .MF ("memo for file"), .EG ("engineer's notes") and .TR (Computing Science Tech. Report).

Headings

```
.NH                 .SH
Introduction.       Appendix I
.PP                 .PP
text text text      text text text
```

1. Introduction
text text text

Appendix I
text text text

A Simple List

```
.IP 1.
J. Pencilpusher and X. Hardwired,
.I
A New Kind of Set Screw,
.R
Proc. IEEE
.B 75
(1976), 23-255.
.IP 2.
H. Nails and R. Irons,
.I
Fasteners for Printed Circuit Boards,
.R
Proc. ASME
.B 23
(1974), 23-24.
.LP (terminates list)
```

1. J. Pencilpusher and X. Hardwired, *A New Kind of Set Screw,* Proc. IEEE **75** (1976), 23-255.
2. H. Nails and R. Irons, *Fasteners for Printed Circuit Boards,* Proc. ASME **23** (1974), 23-24.

Displays

```
text text text text text text
.DS
and now
for something
completely different
.DE
text text text text text text
```

hoboken harrison newark roseville avenue grove street east orange brick church orange highland avenue mountain station south orange maplewood millburn short hills summit new providence

    and now
    for something
    completely different

murray hill berkeley heights gillette stirling millington lyons basking ridge bernardsville far hills peapack gladstone

Options: .DS L: left-adjust; .DS C: line-by-line center; .DS B: make block, then center.

Footnotes

```
Among the most important occupants
of the workbench are the long-nosed pliers.
Without these basic tools*
.FS
* As first shown by Tiger & Leopard
(1975).
.FE
few assemblies could be completed. They may
lack the popular appeal of the sledgehammer
```

Among the most important occupants of the workbench are the long-nosed pliers. Without these basic tools* few assemblies could be completed. They may lack the popular appeal of the sledgehammer

* As first shown by Tiger & Leopard (1975).

Multiple Indents

```
This is ordinary text to point out
the margins of the page.
.IP 1.
First level item
.RS
.IP a)
Second level.
.IP b)
Continued here with another second
level item, but somewhat longer.
.RE
.IP 2.
Return to previous value of the
indenting at this point.
.IP 3.
Another
line.
```

This is ordinary text to point out the margins of the page.

1. First level item
   a) Second level.
   b) Continued here with another second level item, but somewhat longer.
2. Return to previous value of the indenting at this point.
3. Another line.

Keeps

Lines bracketed by the following commands are kept together, and will appear entirely on one page:

.KS	not moved	.KF	may float
.KE	through text	.KE	in text

Double Column

```
.TL
The Declaration of Independence
.2C
.PP
When in the course of human events, it becomes
necessary for one people to dissolve the
political bonds which have connected them with
another, and to assume among the powers of the
earth the separate and equal station to which
the laws of Nature and of Nature's God entitle
them, a decent respect to the opinions of
```

The Declaration of Independence

When in the course of human events, it becomes necessary for one people to dissolve the political bonds which have connected them with another, and to assume among the powers of the earth the separate and equal station to which the laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

We hold these truths to be self-evident, that all men are created equal, that they are endowed by their creator with certain unalienable rights, that among these are life, liberty, and the pursuit of happiness. That to secure these rights, governments are instituted among men,

Equations

```
A displayed equation is marked
with an equation number at the right margin
by adding an argument to the EQ line:
.EQ (1.3)
x sup 2 over a sup 2 ~=~ sqrt {p z sup 2 +qz+r}
.EN
```

A displayed equation is marked with an equation number at the right margin by adding an argument to the EQ line:

$$\frac{x^2}{a^2} = \sqrt{pz^2+qz+r} \qquad (1.3)$$

```
.EQ  I (2.2a)
bold V bar sub nu~=~left [ pile {a above b above
c } right ] + left [ matrix { col { A(11) above .
above . } col { . above . above .} col {. above .
above A(33) }} right ] cdot left [ pile { alpha
above beta above  gamma } right ]
.EN
```

$$\bar{\mathbf{V}}_\nu = \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} A(11) & . & . \\ . & . & . \\ . & . & A(33) \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \qquad (2.2a)$$

```
.EQ  L
F hat ( chi ) ~ mark = ~|del V|sup 2
.EN
.EQ  L
lineup =~ {left ( {partial V} over {partial x} right )
} sup 2 + { left ( {partial V} over {partial y} right
) } sup 2 ~~~~~~ lambda -> inf
.EN
```

$$\hat{F}(\chi) = |\nabla V|^2$$

$$= \left(\frac{\partial V}{\partial x}\right)^2 + \left(\frac{\partial V}{\partial y}\right)^2 \qquad \lambda \to \infty$$

```
$ a dot $,  $ b dotdot$,  $ xi tilde times y vec$:
```

$\dot{a}$, $\ddot{b}$, $\tilde{\xi}\times\vec{y}$.   (with delim $$ on, see panel 3).

See also the equations in the second table, panel 8.

Some Registers You Can Change

Line length
.nr LL 7i

Title length
.nr LT 7i

Point size
.nr PS 9

Vertical spacing
.nr VS 11

Column width
.nr CW 3i

Intercolumn spacing
.nr GW .5i

Margins — head and foot
.nr HM .75i
.nr FM .75i

Paragraph indent
.nr PI 2n

Paragraph spacing
.nr PD 0

Page offset
.nr PO 0.5i

Page heading
.ds CH Appendix (center)
.ds RH 7-25-76 (right)
.ds LH Private (left)

Page footer
.ds CF Draft
.ds LF
.ds RF   similar

Page numbers
.nr % 3

Tables

```
.TS          (ⓣ indicates a tab)
allbox;
c s s
c c c
n n n.
AT&T Common Stock
YearⓣPriceⓣDividend
1971ⓣ41-54ⓣ$2.60
2ⓣ41-54ⓣ2.70
3ⓣ46-55ⓣ2.87
4ⓣ40-53ⓣ3.24
5ⓣ45-52ⓣ3.40
6ⓣ51-59ⓣ.95*
.TE
* (first quarter only)
```

AT&T Common Stock		
Year	Price	Dividend
1971	41-54	$2.60
2	41-54	2.70
3	46-55	2.87
4	40-53	3.24
5	45-52	3.40
6	51-59	.95*

* (first quarter only)

The meanings of the key-letters describing the alignment of each entry are:

c	center	n	numerical
r	right-adjust	a	subcolumn
l	left-adjust	s	spanned

The global table options are center, expand, box, doublebox, allbox, tab (x) and linesize (n).

```
.TS          (with delim $$ on, see panel 3)
doublebox, center;
c c
l l.
NameⓣDefinition
..sp
Gammaⓣ$GAMMA (z) = int sub 0 sup inf \
  t sup {z-1} e sup -t dt$
Sineⓣ$sin (x) = 1 over 2i ( e sup ix - e sup -ix )$
Errorⓣ$ roman erf (z) = 2 over sqrt pi \
  int sub 0 sup z e sup {-t sup 2} dt$
Besselⓣ$ J sub 0 (z) = 1 over pi \
  int sub 0 sup pi cos ( z sin theta ) d theta $
Zetaⓣ$ zeta (s) = \
  sum from k=1 to inf k sup -s ~~( Re~s > 1)$
.TE
```

Name	Definition
Gamma	$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$
Sine	$\sin(x) = \frac{1}{2i}(e^{ix}-e^{-ix})$
Error	$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$
Bessel	$J_0(z) = \frac{1}{\pi}\int_0^\pi \cos(z\sin\theta)d\theta$
Zeta	$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \quad (\mathrm{Re}\ s > 1)$

Usage

Documents with just text:
troff -ms files
With equations only:
eqn files | troff -ms
With tables only:
tbl files | troff -ms
With both tables and equations:
tbl files|eqn|troff -ms

The above generates STARE output on GCOS: replace −st with −ph for typesetter output.

ADDENDUM: THE TEKTRONIX VERSION OF THE *ms* PACKAGE

Rick LeFaivre

Computer Research Laboratory
Applied Research Group
Tektronix Labs

Calling Sequence

ms is a macro package for the *nroff* and *troff* text processing programs. Although it can be invoked by explicitly calling *nroff/troff*, an easier alternative is to use the new *ms* command:

ms *options files*

Available options (which can occur in any order but must precede the file(s)) are listed in Table 1. The most frequently used option is **−T**, which specifies the type of printing device to which the output is being directed. For example, **−Tq** specifies that the output is being directed to a Qume printer, **−Tlpr** sends the output to the Printronix line printer, and **−Tvpr** directs the output to a Versatec printer/plotter as typeset text. Table 2 gives a list of available terminal types. If the **−T** option is not given, a "standard" ASCII teminal is assumed.

Basic Text Formatting

To use *ms*, one prepares a file (using the editor) which contains the text to be formatted and commands to control its appearance. Commands start with a period at the beginning of the line* and consist of one or two uppercase letters, possibly followed by one or more arguments. Certain commands may be embedded directly in the text—in this case they start with a backslash character (\)†. If you need to have this character appear in your printed document, you may type it in using the \e command (echo command character); for example, A\eB prints as A\B.

In the absence of any formatting commands, text appearing in the file will be read, filled to the right margin, adjusted to make the right margin even, and printed. Input lines ending with '.', '?', or '!' will have two spaces appended following the punctuation mark; you must explicitly type the two spaces following a sentence only if it ends in the *middle* of an input line (multiple spaces appearing in the text will be retained). In general, it is recommended that a new line be started (at least) for each sentence in the input text—sentences may then be inserted, deleted, and moved much more easily than if they started in the middle of a line.

Line breaks are made only on word boundaries or at standard hyphenation points within words. If you wish to include spaces in the text which should not be padded or broken across a line, an "unpaddable space" may be entered by typing '\ ' (backslash space). For example, 'New\ York' would be guaranteed to appear on one line with no extra intervening spaces. Hyphenation is normally done automatically—you should *not* hyphenate a word on input unless it is a compound word, as in mother-in-law (in this case make sure the entire sequence is entered on one line so that no intervening spaces will be inserted). If you prefer unhyphenated text, the .HY command may be used to turn off hyphenation*. Similarly, .AD is available to

* For obscure reasons a single quote mark may also be used to start a command—this is of interest because it means an input line may not begin with a single quote.

† Note that the editor *xed* requires that you type two consecutive backslashes to have a single backslash included in your text.

* Hyphenation may be turned off for individual *words* by preceding the word with '\%', as in \%nottobehyphenated. If '\%' appears *inside* of a word it indicates a possible hyphenation point (overriding the

specify whether text should be "adjusted" (the default) or printed with a "ragged" right margin. Commands are also available for "breaking" lines before the margin is reached (.BR), inserting blank lines (.SP), requesting double-spacing (.LS), moving to a new page (.BP) or column (.BC), conditionally moving to a new page if not enough space remains on the current page (.NE), and changing the page number (.PN). All of these commands are described in detail in the reference section of this document.

Overall Format

Manuscript files may start with a simple paragraph (.PP or .LP), a section heading (.SH or .NH), or a title line (.TL). They may also start with an "overall format" command which describes the general format of the document: .RP signifies a released paper (with a title page); .TR or .TM signifies an internal technical report or memo (with a Tek Labs cover sheet complete with confidentiality statement); .IOC signifies an IOC; and .LT signifies a letter. Examples of these various formats are appended to the end of this document.

Indentation

There are several mechanisms for generating indented text in *ms*. An individual paragraph may be indented via the .IP command, with an optional label placed to the left of the paragraph. The margin will be reset at the next paragraph command unless the command .RS is given, which "fixes" the current indentation level until a corresponding .RE is encountered. .IS combines .IP and .RS, and may be used to start an indented section which contains multiple paragraphs; all text up to the next .IE will be indented. Nested .IS/.IE pairs are allowed if several levels of indentation are required. .QP, .QS, and .QE are similar to .IP, .IS, and .IE, but indentation is made from both the left and right margins (as for long quotes). If numbered headings are being used, entire sections may be automatically indented by placing the command '.HS I' at the beginning of the document—each section will then be indented an amount proportional to the level of the section heading. Finally, segments of unformatted text may be indented via the .DS I or .ID commands.

Character Fonts and Underlining

As described in the Bell *ms* manual, the commands .B, .I, and .R may be used to switch between boldface, italic (or underlined), and regular text. An alternate way to change fonts is the in-line command \F (or \f), followed by the font name. For example, one can produce a boldface **word** either via the sequence

```
text . . .
.B word
text . . .
```

or via "text . . . \FBword\FR text . . .". Note that the in-line command gives one the ability to switch fonts between individual characters if desired, with no intervening spaces; for example, *a***R***b* could be produced via \FIa\FBR\FIb\FR. The available fonts are:

R	"Roman" or regular font.
B	**Boldface font.**
I	*Italic font.*
U	Underline alphanumeric characters.
C	Continuous underline (including blanks).
BI	***Boldface italics.***
BU	**Underlined boldface**.
BC	**Continuous underlined boldface**.

standard hyphenation computation), as in st\%rangehyp\%henation.

Combination fonts with two-character font names (BI, BU and BC) are requested by including a left parenthesis before the name: \F(BI. Note that Italics is available only on the Versatec (**—Tvpr**); on other output devices font I is mapped into font U. Similarly, fonts U and C map into font I on the Versatec; to actually underline a word on the Versatec, use the .UL command. Boldface is produced on certain output devices by overprinting; on the Printronix printer, which is incapable of overprinting, font B is mapped into font C. Finally, note that the font is always reset to normal at the beginning of each paragraph. To have multiple paragraphs printed in some font other than Roman, change the string NF (Normal Font) from R to the font desired.

Note that there are a number of special-purpose fonts available for use with the Versatec as replacements for the default R, I, and B fonts. Loading of these special fonts is requested via the .FM (Font Mound) command.

Special Characters

In addition to the normal (ASCII) characters which may be typed directly from the keyboard, there are a number of "special characters" in *ms* which are entered via sequences of the form \(*xx* where *xx* is the name of the character. For example, the "bullet character" ● is produced via the sequence \(bu, and the Greek letter π is produced via \(*p. Table 3 lists the available special characters and how they print on the Versatec—on other output devices an approximation of the characters will be printed, or a blank will be left if no approximation is possible. Table 3 also lists several constructed characters such as 'Λ' and '—' (long dash—entered as *–), as well as various accent marks.

Combinations of overstruck characters which are not available as predefined special characters may be constructed via the *overstrike* command \o'*chars*', which prints *chars* one on top of the other. For example, \o'\(ci\(mu' prints as ⊗. In case you're curious, \(bu on the Qume is the same as \o'o=:–o'.

Superscripts and Subscripts

Superscripts and subscripts may be entered via the sequences *{superscript*} and *[subscript*] (as in X*{2*}, which prints as X^2). These sequences will cause white space to be inserted before or after the line so that the superscript or subscript does not touch the line above or below the current line. On the Versatec the super/subscripts will be printed in a smaller type size than normal. See the *eqn* manual for an alternate way to specify super/subscripting (as well as special characters).

NOTE: Since the superscript and subscript commands require reverse movement of the paper, the command .COL should be placed at the beginning of the file to cause the resultant "reverse linefeeds" to be processed properly when the output is directed to devices without a reverse linefeed capability.

Character Size

Several different character sizes are available when output is directed to the Versatec (**—Tvpr**). The character size is given in "points", with point sizes of 6-12, 14, 16, 18, 20, 22, and 24 currently available (the default is 10 points—Table 3 contains examples of other character sizes). The commands .SZ, .LG and .SM may be used to temporarily change the point size. .SZ takes as an argument either an absolute point size (e.g., '.SZ 16'), or a relative point size change (e.g., '.SZ –3'). .LG is equivalent to '.SZ +2', and .SM is equivalent to '.SZ –2'. As with font switches, there is also an in-line command to change sizes: \s*n* switches to size *n*, and \s±*n* increases or decreases the point size by the specified amount. For example, UNIX is produced via \s–2UNIX\s+2 and a large bullet (●) is produced via \s16\(bu\s0 (\s0 restores the previous point size).

Note that, as with fonts, the character size is automatically reset at the start of each paragraph. To have multiple paragraphs (or an entire document) printed in a size other than 10 points, register PS may be set to the desired size (it will take effect at the next paragraph start command). You should also reset register VS, which specifies the vertical spacing (in points) between the baselines of successive lines. VS is normally set to PS+2 (the default is 12), but may be increased to cause additional space to be inserted between lines. Note how this differs from the '.LS 2' command, which causes a full blank line (of height VS points) to be inserted between each line. The appropriate vertical spacing is set automatically by .SZ, .SM and .LG, but is not changed by \s (which should therefore be used for short in-line sequences only).

Command Descriptions

Following is a complete alphabetic list of the commands available in *ms*. The easiest way to locate the command you need to perform some function is to check the Quick Reference List (Table 4) first, then look the command up here. Optional arguments are enclosed in braces {. . .}, and alternatives are separated by '|'.

.1C	Change back to one column after printing at two columns (see .2C). Also performs a page eject.
.2C	Change to two column output. Use .BC to move to the top of the next column. Use .1C to change back to one column output. Causes reverse linefeeds to be inserted—the command .COL should be placed at the beginning of the file for proper processing.
.AB {no}	Begin an abstract. Should follow the .TL, .AU, and .AI (if any) commands. Text between the .AB and .AE will be collected and printed as an indented block, with the word *ABSTRACT* centered before the text. Type '.AB no' instead of '.AB' to omit the word *ABSTRACT* (as was done in this document).
.AD {0|1}	Set Adjustment. '.AD 0' turns off adjustment of the text (i.e., the text will have a "ragged" right margin). '.AD' or '.AD 1' turns adjustment back on.
.AE	End abstract (see .AB).
.AI	Author's Institution. The following line(s) give the group affiliation of the author or authors specified in the .AU command(s) preceding the .AI. Type as many lines as desired.

.AN *c* {*type*}

Define "auto increment number" with name *c* (a single character—upper case letters are recommended to avoid name conflicts). \*c* will cause 1 to be inserted the first time it is used, 2 the second time, and so on. \n*c* will insert the current value without incrementing the number. For example,

```
.AN F
```

might be used to define an automatic footnote numberer which could be used in combination with the superscript mechanism as follows:

```
. . . blah, blah\*{\*F\*}
.FS \nF.
This is the footnote.
.FE
blah, blah . . .
```

The auto increment number may be reset to 0 at any time by reissuing the .AN command. If *type* is included in the .AN command it specifies an output format other than the standard 1,2,3,...: 'A' specifies the sequence A,B,C,..., 'a' specifies a,b,c,..., 'I' specifies I,II,III,IV,..., and 'i' specifies i,ii,iii,iv,... As another example, a long list with labels of the form a), b), ... could be set up as follows:

```
.AN L a
.IP \*L)
First item ...
.IP \*L)
Second item ...
etc.
```

See also the example given under the command .de. Note that PN is in effect a predefined auto increment number which is incremented each time a new page is started (i.e., it contains the current page number). To cause page numbers to be printed in small roman numerals, one could thus do

```
.AN PN i
```

.AU	The following line specifies the author of this document. For multiple authors, type several successive .AU commands.
.B {*text*}	Print *text* in boldface. If no argument, switch to boldface.
.BC	Begin a new column when in .2C mode. Identical to .BP if only one column.
.BD {0}	Start a centered block display—lines up to the next .DE will be collected and then the entire block will be centered. Identical to .DS B, except the display may extend onto the next page. It will be preceded by a blank line unless the optional argument '0' is present.
.BE	(Also known as .B2) End a section to be enclosed in a box (see .BS).
.BP	Begin a new page. Ignored if already at the top of a page—to force a blank page place an empty display on the page: `.BP` `.DS` `.DE` `.BP`
.BR	Break the current line being collected and start a new line. Used infrequently, since many other commands do an automatic break.
.BS {C}	(Also known as .B1) Start a section of text to be enclosed in a box. The end of the boxed text is indicated via .BE. If the optional argument C is present, the box will be centered on the page. Causes reverse linefeeds to be inserted—the command .COL should be placed at the beginning of the file for proper processing.
.BU {0}	Start a "bullet item", i.e., an indented paragraph marked with a bullet character. Identical to `.IP \(bu 4 2` Precede with a blank line unless the optional argument '0' is present.
.BX *word*	Enclose $\boxed{\textit{word}}$ in a box. See comments under .BS regarding .COL.
.CD {0}	Start a centered display—each line up to the next .DE will be centered relative to the current left and right margins. Identical to .DS C, except the display may extend onto the next page. It will be preceded by a blank line unless the optional argument '0' is present.
.CN	Place the standard Tek Labs confidentiality note at this point in the document. This is done automatically at the bottom of the cover sheet for a technical report (.TR) or memo (.TM). May be redefined (via .de) to cause a different confidentiality note to be placed on cover sheets. Sites other than Tek Labs may wish to completely redo the text by changing the file */usr/lib/tmac/tmac.sconfid.*
.COL	Should be placed as the very first line in the file when the document contains commands which cause reverse movement of the paper (*{, *[, .BX, .BS, .2C). Obviates the need for the **−col** option on the *ms* command line. Ignored when output is being directed to a device like the Qume which has a reverse linefeed capability.

.CS {*n*}

Constant spacing mode. Normally used inside a display to cause characters to be spaced evenly. If *n* is missing and output is being directed to the Versatec (**—Tvpr**), proportional spacing is turned off (i.e., each character is printed with a constant width). If *n* is 0, proportional spacing is turned back on. The "width" is taken from register CS (initially 24—don't ask what the units are), and may be changed to pack or stretch the text. If *n* is non-zero, it is taken as the width in place of CS. Ignored if not **—Tvpr**. See the examples at the end of this document for some constant spaced text.

.DA

Cause the current date to be placed at the bottom of each page (i.e., sets the CF string register to the current date). Note that if you want the date to appear only on draft copies of the document, use the .DR command or the **—DR** option on the *ms* command line.

.DE {0}

End a display (see .DS, .ID, .LD, .CD, .BD). Insert a blank line following the display unless the optional argument '0' is present.

.de *name*

Define a command. This *nroff* command may be used to define a new command with name *name*, consisting of one or two characters. The name should be chosen to avoid conflict with existing *nroff* or *ms* command names—since all *nroff* command names are lower case and all *ms* command names are upper case, a combined upper case/lower case name is always safe. The command definition appears on succeeding lines, terminated by a line with two periods: '..'. All text and commands between the .de and the terminating '..' will be inserted when the command *.name* is given. Note that for obscure reasons any references to in-line (\) commands should contain *two* backslashes when they appear in a command definition*. As a simple example, the following sequence defines a "numbered paragraph" command which produces automatically numbered lists:

```
.de Np
.IP (\\*N)
..
```

The numbering sequence is (re)initialized via a '.AN N' command, so that

```
.AN N
.Np
First item . . .
.Np
Second item . . .
```

would produce:

(1) First item . . .

(2) Second item . . .

For information on defining more complex commands which accept arguments and include conditional sequences, consult the *NROFF/TROFF User's Manual.*

.DR

Insert D R A F T and the current date at the bottom of each page, immediately following the page footer (if any). Normally placed at the beginning of a draft document. Identical to including the **—DR** option on the *ms* command line.

* *xed* users: this means you must type *four* backslashes to get the two inserted into your text!

.DS {I|L|C|B} {*indent*} {0}

Start a display. Lines up to the next .DE will be read and printed unchanged except for their horizontal placement relative to the current left and right margins—L left justifies, C centers each line, B centers the entire block, and I indents 8 spaces (this is the default). If an I is present, the indentation may be changed from 8 spaces by including a number *indent.* A blank line will be inserted preceding the display unless a final '0' is present. If the display will not fit on the current page a .BP will be performed preceding the display to move it to the top of the following page. Displays which may extend over page boundaries may be requested via .ID, .LD, .CD, and .BD.

.ds *name string*

Define a string register. Typically used to redefine one of the strings listed in Table 6, although users may define their own strings if desired (strings with one-character names are subsequently referenced via '\**x*', while two-character names are referenced via '\*(*xy*'). As with .de, any references to in-line (\\) commands require *two* backslashes. For example, chapter-page numbering, with the chapter (in Roman Numerals) and page number within the chapter printed at the bottom of each page, may be set up as follows:

```
.AN C I
.ds CH
.ds CF \\nC–\\n(PN
```

Each chapter would then start with the following sequence (which could, of course, be defined as a new command via .de):

```
.PN 1
.BP
.SH CE
CHAPTER \*C
.PP
```

.EH

End a heading. Necessary only when a heading (.SH or .NH) is to be run in with the following text, as in

```
.NH
Section Heading.
.EH
Note that the heading appears on the same
line as this text.
```

.EN

End an equation. See *eqn* for details.

.EQ {C|I|L} {*label*}

Begin an equation. The optional arguments C, I, and L indicate a centered (default), indented, or left-justified equation. *label* is an optional label to place to the right of the equation. See *eqn* for details.

.FE

End a footnote (see .FS).

.FM {R|I|B} *fontname*

Mount a new font (ignored if not **—Tvpr**). Must appear at the beginning of a document. Only three fonts may be mounted in any single document, with the "Hershey" R, I, and B fonts the default. To switch to another complete font containing roman, italic, and boldface characters, simply type .FM followed by the font name:

```
.FM nonie
```

To access a font without a complete set, choose which standard font (R, I, or B) you want replaced by the chosen font. For example, to use oldenglish in place of boldface, enter

```
.FM B oldenglish
```

\fB would then specify oldenglish. Note that the default Hershey fonts are currently the only fonts with a complete set of point sizes—when referencing an alternate font you must take care to switch to a point size which is available. The currently available fonts are listed in */usr/lib/vfont/list*, with examples in the latest *Font Catalog.*

.FS {*label*}	Begin a footnote. Text between the .FS and .FE will be saved and placed at the bottom of the page. *label* is the footnote label (e.g., *) which is to be placed to the left of the footnote. On the Versatec, footnotes are printed in a smaller type size (i.e., a .SM is done automatically).
.HL {*c*}	Draw a horizontal line across the page from the current left margin to the current right margin. If a character *c* is present it will be used to draw the line instead of the default line drawing character ('_').
.HS {O|I}	Specify heading style (normally placed at the beginning of the document to affect the overall appearance). An argument of 'O' specifies that numbered headings (.NH command) should be printed in outline form (I., A., 1., a., i.) instead of normal section form (1., 1.1, 1.1.1.). An argument of 'I' specifies that numbered sections should be automatically indented an amount proportional to the heading level. The default amount to indent each section is 4 spaces (the value of number register NI). If both 'O' and 'I' are desired, they may be given in either order (separated by a space).
.HY {0|1}	Set hyphenation. '.HY 0' turns off automatic hyphenation, while '.HY' or '.HY 1' turns it back on.
.I {*text*}	Print *text* in italics if **−Tvpr**, underline otherwise. If no argument switch to italics/underlining.
.ID {*in*} {0}	Start an indented display—each line up to the next .DE will be printed "as is" indented 8 spaces (or *in* spaces if present). Identical to .DS I, except the display may extend onto the next page. It will be preceded by a blank line unless '0' is present.
.IE {0}	End an indented section. Identical to .RE followed by .LP, with the left margin restored to its value before the last .IS. Follow with a blank line unless '0' is present.
.IOC {H}	Start an IOC. If H is present a fake IOC header will be printed (suitable for informal memos within the group). Otherwise it is assumed that the special IOC paper next to the Qume will be used and the preprinted header is skipped. The current date (which may be changed via .ND) will be printed along with information obtained from the following commands:

.TO The person to whom the IOC is being sent. If there are several recipients, several .TO commands may be used (up to 5). Alternately, up to 5 lines of information may be associated with a single individual:

```
.TO A. R. Researcher, 50-543
.TO Manager, Research Research
.TO Applied Research Group
```

To create a "distribution list" type .TO by itself and follow it with the recipients, one per line (a delivery station may also be added, with tabs used to line things up if desired). The word "Distribution" will then be placed in the 'To:' field, and a distribution list (in two columns if necessary) will be placed at the bottom of the page if it fits, or on a blank page suitable for stapling to the front of the IOC if it doesn't fit or if this is a multi-page memo. Note that distribution lists kept in separate files may easily be included via the .RD command. The word "Distribution" in the 'To:' field may be changed by setting the string register DI:

```
.ds DI AR Staff
.TO
.RD arstaff.names
```

.FR The person sending the IOC. If there are several senders, several .FR commands may be used (up to 5). Alternately, up to 5 lines of information may be associated with a single individual, as with the .TO command.

.CC Individuals to whom copies are to be sent (optional). Enter names as with .TO (i.e., up to 5 .CC commands, or .CC followed by names to be placed at the end of the IOC).

.SU The subject of the memo. Will also appear in the page header on multi-page memos.

.DA If the IOC is announcing a meeting, this is the date of the meeting (optional).

.TI Ditto for the time.

.PL Ditto for the place.

These commands should be followed by an .LP, .PP, .SH, or .NH to start the text of the IOC. See the examples at the end of this document for some sample IOCs.

.IP {*lab* {*in* {*lin* {*hang*}}}} {0}

Start an indented paragraph. The text will be indented *in* spaces, the label *lab* will be indented *lin* spaces, and the first line of text will be indented *hang* spaces from the rest of the text (*hang* may be negative to cause a "hanging indent"). Pictorially:

```
|Original Left Margin
|
|---lin--→lab        |---hang--→First line of paragraph
|                    |second line of paragraph . . .
|                    |
|---------in--------→|New Left Margin
```

The default label is "" (the null string). The default indents are 5 (the value of number register PI), 0, and 0 unless the immediately preceding paragraph was also started with .IP, in which case the values specified there are used. Note that if *lab* contains any blanks, they should be entered as unpaddable spaces, e.g., long\ label. The paragraph will be preceded by a blank line unless a final '0' is present.

.IS {*in*} {0} Start an indented section. Similar to a .IP (without a label) followed by a .RS—all text up to the next .IE command will be indented *in* spaces. If *in* is missing, the value of number register PI is used (initially 5). Precede with a blank line unless '0' is present.

.JU '*left*'*center*'*right*'

Justify a single line. Print a line of text consisting of *left* at the left margin, *center* centered, and *right* at the right margin. Any of the three fields may be empty, so to right justify some text type

```
.JU '''Right justify this'
```

If any of the fields contain sequences of multiple spaces, the entire string should be enclosed in quotes:

```
.JU "'left'left-center    right-center'right'"
```

.KE End a keep (see .KS and .KF).

.KF Start a floating keep. All text up to the next .KE will be collected and printed if it fits on the current page, or "floated" through the text to the top of the next page if it doesn't. .HL is useful for setting floating keeps off from the rest of the text.

.KS Start a regular keep. All text up to the next .KE will be collected and printed, preceded by a page eject if it won't fit on the current page. .DS does an automatic .KS, so it is rarely used directly.

.LD {0} Start a left-justified display—each line up to the next .DE will be printed "as is". Identical to .DS L, except the display may extend onto the next page. It will be preceded by a blank line unless '0' is present.

.LG Make type size larger (ignored if not **−Tvpr**). Equivalent to '.SZ +2'. The larger type size will remain in effect until a .SZ, .SM or .NL command is encountered, or until a new paragraph is started. Note that a .LG is done automatically for .TL titles.

.LP {0} Start a left-blocked paragraph. Precede with a blank line unless '0' is present.

.LS *n* Set the line spacing. The initial mode is single-spacing—do a '.LS 2' to switch to double-spacing, and a '.LS 1' to switch back to single-spacing.

.LT {*in*} Start a letter. Skips over the Tek letterhead, prints the current date, and accepts an address and salutation. Start the actual text with a .PP or .LP. *in* gives the amount to indent the date—0 causes it to appear at the left margin, while a missing *in* causes it to appear right-justified at the right margin. See the example at the end of this document.

.ND *date* Set a new date if you want to fake the system out.

.NE *n* Begin a new page if less than *n* lines remain on the current page (NE stands for "NEed *n* lines"). As with .SP, *n* may be in inches (1i) or centimeters (3c) if desired. Done automatically for displays and keeps.

.NH {*level*} {*font*} {CE}

Print a numbered heading at level *level* (the default is 1, i.e., a top-level heading), preceded by a blank line. If CE is present the heading will be centered. If a *font* is present the heading will be printed in that font, otherwise the font name in the string register HF will be used (initially **B**). Similarly, register HS may be set to cause headings to be printed in a different point size on the Versatec. The text following the heading will be automatically indented an appropriate amount if the '.HS I' command has been given. The heading number will be printed in "outline" form if the '.HS O' command has been given (see .HS). The heading should follow the .NH command and should normally be terminated by a paragraph start command—if you want the heading to be run in with the following text, however, end it with an .EH command.

.NL	Return type size to normal (cancels effect of .SZ, .LG or .SM).
.nr *name val*	Set number register. Typically used to redefine one of the number registers listed in Table 6.
.P1	Print page one header. The header at the top of page one is normally suppressed. To cause it to be printed, include the command .P1 at the start of the document.
.PN *n*	Set the page number of the next page to *n*. Note that the format in which the page number is printed may be changed from the normal arabic numerals if desired (see the .AN command).
.PP {0}	Start a normal paragraph with the first line indented 5 spaces (the value of number register PI). Precede with a blank line unless '0' is present.
.QE {0}	End a quoted section (see .QS). Insert a blank line unless '0' is present.
.QP {0}	Start a quoted paragraph which is indented 5 spaces (the value of number register QI) from both the left and right margins. Precede with a blank line unless '0' is present.
.QS {0}	Start a quoted section, with both margins moved in 5 spaces (the value of number register QI). These margins will remain in effect until the next .QE. Precede with a blank line unless '0' is present.
.R	Return to regular (Roman) font.
.RD {*file*}	Read input from *file* just as if it were included in the text. If *file* is missing, read from the standard input until a blank line is encountered.
.RE	End a relative indented section (see .RS).
.RP	Start a released paper with a title page. Same format as a .TR but without a cover sheet. This document was printed with .RP.
.RS	Start a relative indented section. Should follow an indented paragraph to cause succeeding paragraph commands (.LP, .IP, etc.) to be relative to the current left margin. This margin will remain in effect until the next .RE.
.SE	End a sorted section (see .SO).

.SH {*level*} {*font*} {CE}

Print an unnumbered heading using font *font* (if missing use the font name in the string register HF—initially B), preceded by a blank line. If register HS is set, it gives the point size in which the heading should be printed on the Versatec. If CE is present the heading will be centered. *level* (see .NH) is ignored, except that it indicates the indentation level in the table of contents for a following .TC command. The heading should follow the .SH command and should normally be terminated by a paragraph start command—if you want the heading to be run in with the following text, however, end it with an .EH command.

.SM

Make type size smaller (ignored if not **−Tvpr**). Equivalent to '.SZ -2'. The smaller type size will remain in effect until a .SZ, .LG or .NL is encountered, or until a new paragraph is started. Note that a .SM is automatically performed for footnotes, superscripts, and subscripts.

.SO {*flags*}

Sort everything up to the next .SE. *flags* are options which will be passed to the UNIX *sort* routine. Of special interest is +*n*, which skips *n* fields of non-blank characters—a flag of +1 says to sort on the second field, and might be used to sort a list of names typed as first name last name. For example, consider the following IOC distribution list:

```
.TO
.SO +1
Sherlock Holmes  50-123
Sam Spade        50-321
Peter Wimsey     50-213
Jane Marple      50-312
.SE
```

'.SO +2' could have been used instead to sort by delivery station.

.SP {*n*}

Insert *n* blank lines (or the specified amount of blank space if *n* is in inches (1i) or centimeters (3c)). If this would go past a page boundary, do a .BP instead. If *n* is missing the value of register PD is used for the distance to space. .SP commands which happen to fall at the beginning of a page are ignored—if you really want to insure that the blank lines appear in the text (for example to reserve room for a figure) place the .SP inside of a .DS/.DE pair.

.SZ *n*

Set the current type size to *n* points (ignored if not **−Tvpr**). If *n* is preceded by a '+' or '−', increment or decrement the current point size by the specified amount. The new point size will remain in effect until a .SZ, .SM, .LG or .NL is encountered, or until a new paragraph is started.

.TA *t1 t2* ...

Set tabs. Use inside a display to set tab stops for column alignment. Because of the proportional spacing on the Versatec, tab stops should be used to line up text (or .CS mode should be entered). The stops may be entered as character positions (8n 16n ...), inches (0.5i 1i ...), or centimeters (1c 2c ...). In addition to normal tab stops, which cause the text following the tab to be left-justified at the stop, one may also specify right-justifying or centering tab stops by following the tab stop with an **R** or **C**. For example,

```
.DS B
.TA 2iC 4iR
\FIItemⓉManufacturerⓉCost\FR
.SP
BaublesⓉPenneysⓉ$5.98
BanglesⓉSearsⓉ$234.98
BeadsⓉMontgomery WardsⓉ$49.95
.DE
```

(where Ⓣ indicates a TAB character) generates the following:

```
Item              Manufacturer              Cost

Baubles             Penneys                $5.98
Bangles              Sears               $234.98
Beads          Montgomery Wards           $49.95
```

Note that tab stops are set and not restored by several *ms* commands (e.g.,

.IP), so you should reset them for any display in which you wish to use tabs.

.TC *text* — Place *text* in the table of contents and also include it in the text. If *text* contains sequences of multiple spaces, it should be enclosed in quotes: "*text*". Normally used after a .SH or .NH command to cause a section heading to be placed in the table of contents:

```
.NH 1
.TC This is the Section Heading
.PP
```

If used following a .NH the section number will be included in the contents; for both .NH and .SH the section number argument is used to indicate the indentation level within the table of contents. A blank line will precede each top-level heading; additional blank lines may be inserted by issuing a .TC with no arguments. The table of contents will be preceded by the word *CONTENTS*, centered in italics. To change this heading you may redefine the .PC command, which is invoked at the top of the page. The initial definition of .PC is:

```
.de PC
.SH I CE
CONTENTS
.LP
.SP 1
..
```

Note that the table of contents will be printed out at the end of the document, and must be physically moved to the front.

.TE — End a table. See *tbl* for details.

.TH — End heading section of table. See *tbl* for details.

.TL — The title of this document follows. It will be printed centered in boldface (in a larger type size if **—Tvpr**). Note that a .BR command may be used to force a multi-line title.

.TM {*number*} — Start a technical memo. *number* is the optional memo number to be printed on the cover sheet.

.TR {*number*} — Start a technical report. *number* is the optional report number to be printed on the cover sheet. See the example at the end of this document.

.TS {H} — Start a table. If H is present the table has a repeated heading. See *tbl* for details.

.UL *word* {*x*} — Underline *word*, even if **—Tvpr**. Any spaces appearing in *word* should be unpaddable, e.g., long\ word. If *x* is present, it is appended immediately after *word* with no intervening spaces.

.XN *text* — Add *text* to the index without a page number. Useful for cross references such as

```
.XN Twain, Mark (See Clemens, Samuel)
```

.XX *text* — Add *text* to the index with the current page number. For example:

```
Text may be indented
.XX Indentation
by using the .IP and .IS commands.
```

The index will be automatically sorted and printed at the end of the document (but before the table of contents, if any). It will be preceded by the word *INDEX*, centered in italics. To change this heading or perform any other initializations, you may redefine the .PX command, which is invoked at the top of the page. The initial definition of .PX is:

```
.de PX
.SH I CE
INDEX
.LP
.SP 1
..
```

To change things, for example, to insert INDEX into the table of contents and cause the index to be printed in two columns using a smaller type size, one would do:

```
.de PX
.SH I CE
.TC INDEX
.LP
.SP 1
.2C
.SM
..
```

This is the definition of .PX used in this document.

Table 1 — *ms* Options

The following options may be included on the *ms* command line:

—eqn	The *eqn* (*neqn* for *nroff*) preprocessor is needed (.EQ used).
—tbl	The *tbl* preprocessor is needed (.TS used).
—col	The *col* post processor is needed (.2C, .BS, or some other command which produces reverse motion was used on a device which doesn't have a reverse linefeed). *col* may be invoked automatically when needed by inserting the command .COL at the beginning of the file.
—T*type*	Set output characteristics for device of type *type* (see Table 2 or the file */usr/lib/nterm/list* for a list of available devices). If *type* indicates a line printer (e.g., **—Tlpr**), output will be printed via *lpr*. If it indicates the Versatec printer (**—Tvpr**), *troff* will be invoked and the typeset output will be printed via *vcat*. If a Tektronix 4014 is indicated (**—T4014**), *troff* output will be piped through the typesetter simulator *tc*.
——lpr	Don't send to *lpr* even though **—Tlpr** specified. Used, for example, to dump a line printer listing to a file. Must follow **—Tlpr** option.
——vpr	Don't send to *vcat* even though **—Tvpr** specified. Allows Versatec output to be previewed (in rough form) to check line and page breaks. Must follow **—Tvpr** option.
—s	Stop at the start of each page after the first. Hit LINEFEED to continue after positioning paper.
—o*list*	Output only those pages whose page numbers appear in *list*, which consists of numbers (5) and number ranges (5-7) separated by commas.
—DR	Insert D R A F T and the current date at the bottom of each page. Same as including the .DR command in the text.
—rL*n*	Set the page length to *n* lines (66), inches (11i), or centimeters (28c). Default is 11 inches.
—rW*n*	Set the line width to *n* characters (72), inches (6.5i), or centimeters (20c). Default is 6 inches for a paper or letter, 6.5 inches for an IOC. Same as setting register LL in the text.
—rO*n*	Offset each page *n* characters (5), inches (0.5i), or centimeters (2c) to the right. Same as setting register PO in the text.
—12	Force wide line on 10-pitch device. Output on the Qume is normally printed at 12 chars/inch, while output to the printer or terminal is at 10 chars/inch. This option forces output to a 10-pitch device to be formatted for 12 chars/inch so that the number of lines printed will be the same as on the Qume.

Table 2 — Available −T Switches

The following terminal type options are available at Tektronix for use with *ms* (and *nroff*). Consult */usr/lib/nterm/list* for an on-line (and possibly more up-to-date) list.

−Tascii	This is the default terminal type. It specifies a standard ascii terminal with spacing at 10 chars/inch and 6 lines/inch. The only non-standard feature is the use of ESC 7 for reverse linefeeds (these can be caught by the *col* filter). As with all of the following, the printed line width is 6 inches (6.5 for an IOC) and the page length is 11 inches.
−Tq	This specifies a Qume SPRINT-5 printer with spacing at 12 chars/inch and 6 lines/inch. The left margin should be set at 16 spaces from the edge of the paper (13 for an IOC), and the paper should be positioned so the top red line on the plastic paper guide is even with the top of the paper. Various special characters will be produced by going into plot mode—if they are used make sure the paper release lever is back (to engage the platen) or the small vertical movements will be lost. The special characters are optimized for the standard elite 12 print wheel.
−Tq−10	This specifies a Qume with spacing at 10 chars/inch and 6 lines/inch. In this case you may wish to use the pica 10 printwheel instead of the standard elite 12.
−Tq−8	This specifies a Qume with spacing at 12 chars/inch and 8 lines/inch.
−Tq−lg	Identical to **−Tq**, except the special characters are optimized for the letter gothic 12 (wp) print wheel. Because several characters are missing on this print wheel, some of the special characters available with the elite 12 print wheel will print as blanks instead.
−Tq−lg−10	Identical to **−Tq−10**, except optimized for the letter gothic print wheel.
−Tq−lg−8	Identical to **−Tq−8**, except optimized for the letter gothic print wheel.
−Td	This specifies a Diablo HyType II printer, with spacing at 12 chars/inch and 6 lines/inch. Same as **−T450−12**.
−Td−10	This specifies a Diablo HyType II printer, with spacing at 10 chars/inch and 6 lines/inch. Same as **−T450**.
−Td−8	This specifies a Diablo HyType II printer, with spacing at 12 chars/inch and 8 lines/inch. Same as **−T450−12−8**.
−T4025	This specifies a Tek 4025 terminal. The line width and page length will be set to values compatible with the hardcopy unit, and output directed to the workspace will have actual underlined and boldface ("enhanced") characters. Note that an assumption is made that the command character is set to '^_' (control underscore).
−Tvt100	This specifies a DEC VT-100 terminal with the advanced video option. Underlining, boldface, and several special characters will be handled correctly.
−Tpt100	Identical to **−Tvt100**—refers to a Plessey PT-100.

—Tlpr	This specifies a Printronix line printer. On the ARG system, output will be directed to the printer on the third floor of Bldg. 50. Since the Printronix has no overprinting capability, boldface text will be underlined.
—Tlpr—t	Identical to **—Tlpr**, except the output will appear in large letters printed at 3 tall lines/inch.
—Tlpr—8	Identical to **—Tlpr**, except vertical spacing will be at 8 lines/inch instead of the normal 6.
—Tlpr—t—8	Identical to **—Tlpr—t**, except large letters will be printed with spacing set to 8 normal lines/inch, giving an effective vertical spacing of 4 tall lines/inch.
—Tupr	This specifies a "standard" line printer. On the ARG system, output will be directed to the printer in the computer room on the fourth floor of Bldg. 50.
—Tvpr	This specifies a Versatec printer/plotter, and uses *troff, trot,* and *vcat* to simulate typeset output. On the ARG system, output will be sent to the Versatec in the computer room on the fourth floor of Bldg. 50.
—Tvpr—t*n*	Uses a scratch tape mounted on tape drive mt*n* as an intermediate storage area for a rasterized image, and then prints the typeset output on the Versatec. This will normally produce a cleaner copy than driving the Versatec directly via **—Tvpr**. The appropriate procedure is to ask the operator to mount a scratch 2400′ tape on an available drive (0 or 1), assign the drive via ass mt*n* and issue the *ms* **—Tvpr—t*n*** command (usually as a background task, since it can take awhile). When the *ms* command is finished, be sure to deassign the drive via deass mt*n* If *n* is missing (**—Tvpr—t**), the raster output is simply directed to the standard output.
—T4014	This specifies a Tek 4014, and uses *troff* and *tc* to simulate typeset output.

Table 3 — Character Set

This table lists all characters currently available in *ms.* The following ASCII characters may be typed in directly:

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

1 2 3 4 5 6 7 8 9 0 ! @ # $ % ^ & * ( ) . , ; : / ? " { } [ ] ~ | _ + =

The following special characters are listed with the printed character first, the input name next, and an explanatory name last.

α	\(*a	alpha	Ο	\(*O	Omicron
β	\(*b	beta	Π	\(*P	Pi
γ	\(*g	gamma	Ρ	\(*R	Rho
δ	\(*d	delta	Σ	\(*S	Sigma
ε	\(*e	epsilon	Τ	\(*T	Tau
ζ	\(*z	zeta	Υ	\(*U	Upsilon
η	\(*y	eta	Φ	\(*F	Phi
θ	\(*h	theta	Χ	\(*X	Chi
ι	\(*i	iota	Ψ	\(*Q	Psi
κ	\(*k	kappa	Ω	\(*W	Omega
λ	\(*l	lambda	\	\e	back slash
μ	\(*m	mu	’	´	close quote
ν	\(*n	nu	‘	`	open quote
ξ	\(*c	xi	´	\(aa	acute accent (or \´)
ο	\(*o	omicron	`	\(ga	grave accent (or \`)
π	\(*p	pi	-	\(hy	hyphen (or -)
ρ	\(*r	rho	−	\(mi	math minus (or \-)
σ	\(*s	sigma	—	\(--	short dash
ς	\(ts	terminal sigma	∗	\(**	math star
τ	\(*t	tau	×	\(mu	multiply
υ	\(*u	upsilon	÷	\(di	divide
φ	\(*f	phi	●	\(bu	bullet
χ	\(*x	chi	□	\(sq	square
ψ	\(*q	psi	○	\(ci	circle
ω	\(*w	omega	°	\(de	degree
Α	\(*A	Alpha	_	\(ru	rule
Β	\(*B	Beta	¼	\(14	1/4
Γ	\(*G	Gamma	½	\(12	1/2
Δ	\(*D	Delta	¾	\(34	3/4
Ε	\(*E	Epsilon	†	\(dg	dagger
Ζ	\(*Z	Zeta	‡	\(dd	double dagger
Η	\(*Y	Eta	′	\(fm	foot mark
Θ	\(*H	Theta	§	\(sc	section
Ι	\(*I	Iota	¢	\(ct	cent sign
Κ	\(*K	Kappa	®	\(rg	registered
Λ	\(*L	Lambda	©	\(co	copyright
Μ	\(*M	Mu	/	\(sl	alternate slash
Ν	\(*N	Nu	√	\(sr	square root
Ξ	\(*C	Xi	‾	\(rn	root en extender

≥	\(>=	>=
≤	\(<=	<=
≡	\(==	identically equal
≠	\(!=	not equal
≃	\(~=	approx =
~	\(ap	approximates
→	\(->	right arrow
←	\(<-	left arrow
↑	\(ua	up arrow
↓	\(da	down arrow
±	\(+-	plus-minus
∪	\(cu	cup (union)
∩	\(ca	cap (intersection)
⊂	\(sb	subset of
⊃	\(sp	superset of
⊆	\(ib	improper subset
⊇	\(ip	improper superset
∞	\(if	infinity
∂	\(pd	partial derivative
∇	\(gr	gradient
¬	\(no	not
∫	\(is	integral sign
∝	\(pt	proportional to
∅	\(es	empty set
∈	\(mo	member of
|	\(br	box vertical rule
☛	\(rh	right hand
☚	\(lh	left hand
🔔	\(bs	bell symbol
|	\(or	or
⎧	\(lt	left top of big curly bracket
⎩	\(lb	left bottom
⎫	\(rt	right top
⎭	\(rb	right bottem
⎨	\(lk	left center
⎬	\(rk	right center
┃	\(bv	bold vertical
⌊	\(lf	left floor (left bottom of big square bracket)
⌋	\(rf	right floor (right bottom)
⌈	\(lc	left ceiling (left top)
⌉	\(rc	right ceiling (right top)
—	*-	long dash
✱	**	big star
"	*(``	open doublequote
"	*(''	close doublequote
≅	*(~=	approx. equal
≐	*(.=	dot equal
∨	*(lo	logical or
∧	*(la	logical and
∴	*(tf	therefore
Å	*(ag	Angstrom
é	*'e	acute accent mark
è	*`e	grave accent mark

ü	*:u	umlat
ê	*^e	caret
ã	*~a	tilde
ě	*Ce	Czech v
ç	*,c	cedilla

Available Point Sizes:

Point Size 6
Point Size 7
Point Size 8
Point Size 9
Point Size 10
Point Size 11
Point Size 12
Point Size 14
Point Size 16
Point Size 18
Point Size 20
Point Size 22
Point Size 24

Table 4 — Quick Reference: Commands

Overall Format	
.RP	Released Paper
.TR	Technical Report
.TM	Technical Memo
.LT	Letter
.IOC	Inter-Office Commun.
.TO	To
.FR	From
.SU	Subject
.CC	Copy
.DA	Date (of Meeting)
.TI	Time (of Meeting)
.PL	Place (of Meeting)
.DR	Draft Version
.COL	Handle Reverse Linefeeds
Title Pages and Headers	
.TL	Title
.AU	Author
.AI	Author's Institution
.AB	Abstract Begin
.AE	Abstract End
.SH	Section Heading
.NH	Numbered Heading
.EH	End Heading
.HS	Specify Heading Style
Basic Text Formatting	
.BR	Break Line
.SP	Insert Blank Space
.JU	Justify Line
.LS	Set Line Spacing
.AD	Set Adjustment
.HY	Set Hyphenation
Character Fonts and Sizes	
.I	Italic Font
.B	Boldface Font
.R	Roman Font
.UL	Underline Word
.SZ	Set Type Size
.SM	Smaller Type Size
.LG	Larger Type Size
.NL	Normal Type Size
.CS	Constant Width Chars
.FM	Mount New Font
Paragraphs and Indentation	
.PP	Paragraph
.LP	Left-Blocked Paragraph
.IP	Indented Paragraph
.RS	Start Relative Indent
.RE	End Relative Indent
.IS	Start Indented Section
.IE	End Indented Section
.QP	Quoted Paragraph
.QS	Start Quoted Section
.QE	End Quoted Section
.BU	Bullet Item
Pages and Columns	
.BP	Begin New Page
.NE	Need Minimum Space
.PN	Set Page Number
.P1	Include Header on Page 1
.DA	Stick Date in Footer
.2C	Switch to Two Columns
.1C	Switch to One Column
.BC	Begin New Column
Displays and Keeps	
.DS	Start Display
.LD	Left-Aligned Display
.ID	Indented Display
.CD	Centered Display
.BD	Centered block display
.DE	End Display
.KS	Start Keep
.KF	Start Floating Keep
.KE	End Keep
Footnotes, Index, Table of Cont.	
.FS	Start Footnote
.FE	End Footnote
.XX	Place in Index
.XN	.XX with No Page Number
.PX	Print Index Header
.TC	Place in Table of Contents
.PC	Print Table of Cont. Header
Equations and Tables (eqn/tbl)	
.EQ	Start Equation
.EN	End Equation
.TS	Start Table
.TE	End Table
.TH	End Table Heading
Miscellaneous	
.BX	Box a Word
.BS	Start Boxed Section (.B1)
.BE	End Boxed Section (.B2)
.HL	Draw Horiz. Line
.TA	Set Tabs
.ND	Set New Date
.RD	Read External Text
.SO	Start Sorted Section
.SE	End Sorted Section
.AN	Define Auto Number
.CN	Tek Labs Confid. Note

Table 5 — Quick Reference: In-Line Commands

Note that in-line commands appearing in command and string register definitions (.de and .ds) should be entered with *two* backslashes (\\).

\space	Unpaddable Space Character
\e	Echo Backslash Character
\%	Suppress Hyphenation
\F*x*	Switch to Font *x* (Also \f)
\F(*xy*	Switch to Font *xy* (Also \f)
\s*n*	Set Character Size to *n* Points
\s±*n*	Increase/Decrease Size by *n* Points
\(*xy*	Special Character *xy*
\o′...′	Overstrike Characters
\"	Ignore Rest of Input Line (For Comments)
*{	Start Superscript
*}	End Superscript
*[	Start Subscript
*]	End Subscript
**x*	Increment and Print Auto Number *x*
\n*x*	Print Auto Number *x* (no incr.)
*(DT	Today's Date
*(DY	Today's Date (Changeable via .ND)
*(DW	Day of the Week
\n(PN	Current Page Number

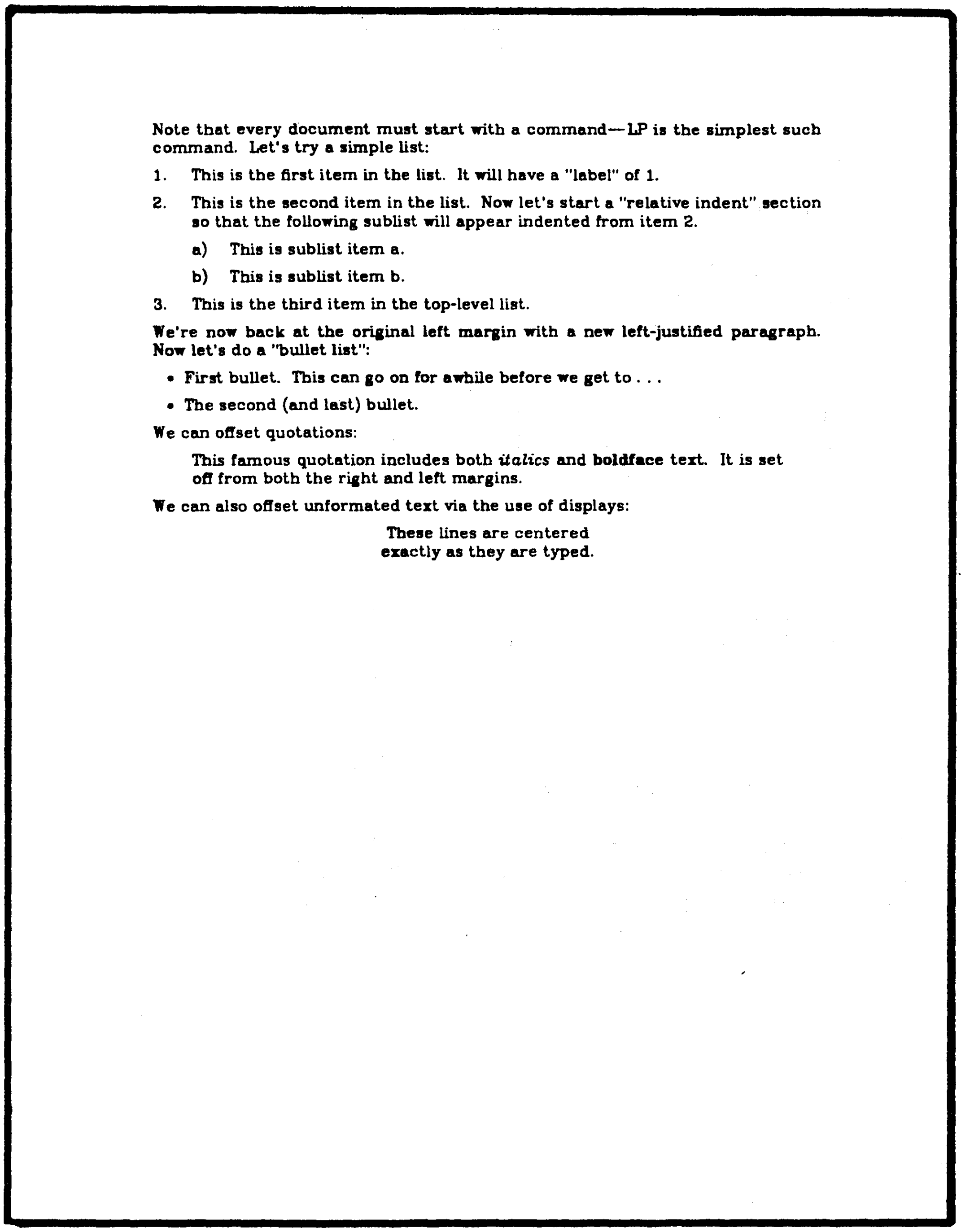
Table 6 — Quick Reference: String and Number Registers

The following are the initial definitions of some of the available string and number registers—they may be changed if desired via the .ds and .nr commands. Numbers should be entered with an appended "scale indicator" specifying how the numbers are to be interpreted: **n** specifies character positions; **v** specifies vertical lines; **i** specifies inches; and **c** specifies centimeters.

.ds LH	Left Portion of Page Header (Initially Null)
.ds CH - \\n(PN -	Center Portion of Page Header
.ds RH	Right Portion of Page Header
.ds LF	Left Portion of Page Footer
.ds CF	Center Portion of Page Footer (\\*(DY if .DA)
.ds RF	Right Portion of Page Footer
.ds NF R	Normal Text Font
.ds HF B	Heading Font (.SH/.NH)
.ds PD 1v	Paragraph Separation (.PP/.DS/.SP—0.5v if **—Tvpr**)
.ds DI Distribution	Default for Missing .TO Argument in IOC
.nr HS 0	Heading Size in Points (0 means no change)
.nr LL 6i	Line Length (6.5i for IOC)
.nr LT 6i	Header/Footer Length (6.5i for IOC)
.nr FL 6i-3n	Footnote Line Length
.nr PO 0	Page Offset (Appropriate Value if **—Tvpr**)
.nr HM 1i	Top Margin (Header in Middle of Margin)
.nr FM 1i	Bottom Margin (Footer in Middle of Margin)
.nr PI 5n	Paragraph (.PP/.IP/.IS) Indent
.nr QI 5n	Quoted Section (.QP/.QS) Indent
.nr NI 4n	Auto Indent for Numbered Sections (.HS I)
.nr PS 10	Character Point Size (Range 6 to 24)
.nr VS 12	Vertical Spacing (Normally PS+2)
.nr CS 24	Constant spacing character width (.CS)

Example — A Simple Document

```
.LP
Note that every document must start with a command\*—LP
is the simplest such command.
Let's try a simple list:
.IP 1.
This is the first item in the list.
It will have a "label" of 1.
.IP 2.
This is the second item in the list.
Now let's start a "relative indent" section so that the
following sublist will appear indented from item 2.
.RS
.IP a)
This is sublist item a.
.IP b)
This is sublist item b.
.RE
.IP 3.
This is the third item in the top-level list.
.LP
We're now back at the original left margin with a new
left-justified paragraph.
Now let's do a "bullet list":
.BU
First bullet.
This can go on for awhile before we get to . . .
.BU
The second (and last) bullet.
.LP
We can offset quotations:
.QP
This famous quotation includes both \FIitalics\FR and
\FBboldface\FR text.
It is set off from both the right and left margins.
.LP
We can also offset unformatted text via the use of
displays:
.DS C
These lines are centered
exactly as they are typed.
.DE
```



Note that every document must start with a command—LP is the simplest such command. Let's try a simple list:

1. This is the first item in the list. It will have a "label" of 1.
2. This is the second item in the list. Now let's start a "relative indent" section so that the following sublist will appear indented from item 2.
   a) This is sublist item a.
   b) This is sublist item b.
3. This is the third item in the top-level list.

We're now back at the original left margin with a new left-justified paragraph. Now let's do a "bullet list":

- First bullet. This can go on for awhile before we get to . . .
- The second (and last) bullet.

We can offset quotations:

> This famous quotation includes both *italics* and **boldface** text. It is set off from both the right and left margins.

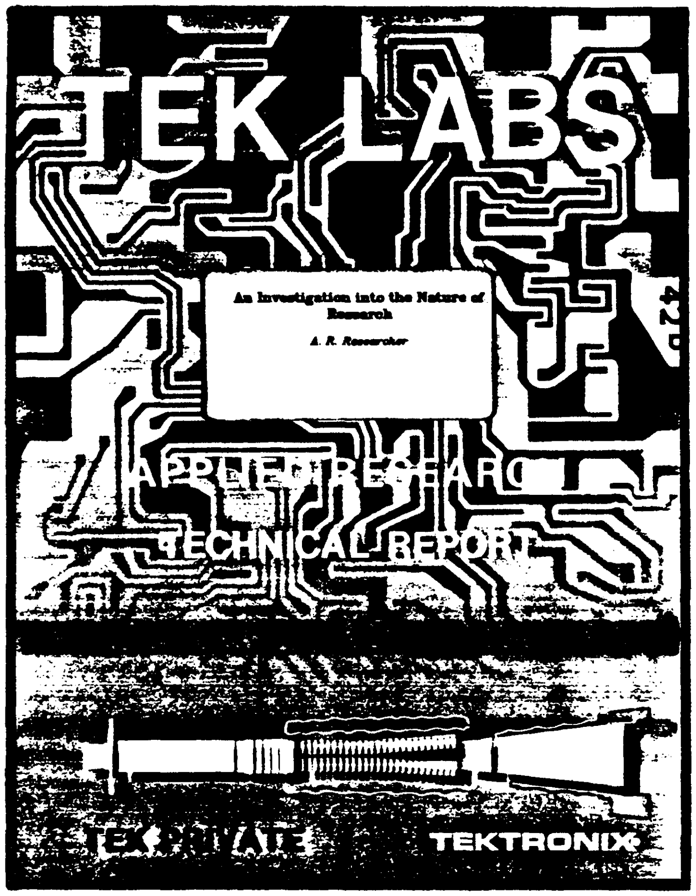
We can also offset unformatted text via the use of displays:

```
                    These lines are centered
                   exactly as they are typed.
```

Example — A Technical Report

```
.TR
.TL
An Investigation into the Nature of Research
.AU
A. R. Researcher
.AI
Research Research Group
Applied Research Group
Tektronix Labs
.AB
This report contains a detailed analysis of a recently
completed five-year research program into the nature of
research carried out by the ARG Research Research Group.
It is a prelude to a newly-formed research project*
.FS *
See "A Long-Term Plan to Research Research Research" by
A. R. Researcher, et al.
.FE
to investigate the potential impact of research research
on ARG research programs.
.AE
.SH
Background
.PP
Research Research is a growing and dynamic discipline . . .
```

The .TR will produce a cover sheet with a technical report confidentiality statement placed automatically at the bottom of the page. Note that the .TR could be replaced with .RP to cause the technical report cover sheet and title page to be replaced with a released paper title page, or could be removed altogether if no cover page is desired.



TEK LABS

An Investigation into the Nature of Research

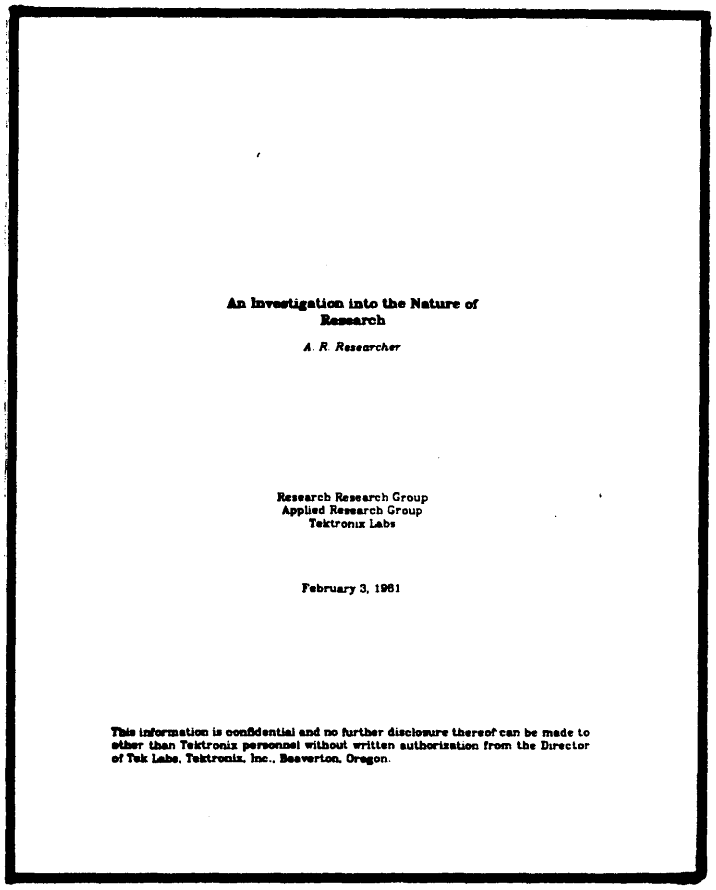
A. R. Researcher

APPLIED RESEARC

TECHNICAL REPORT

TEK PRIVATE

TEKTRONIX



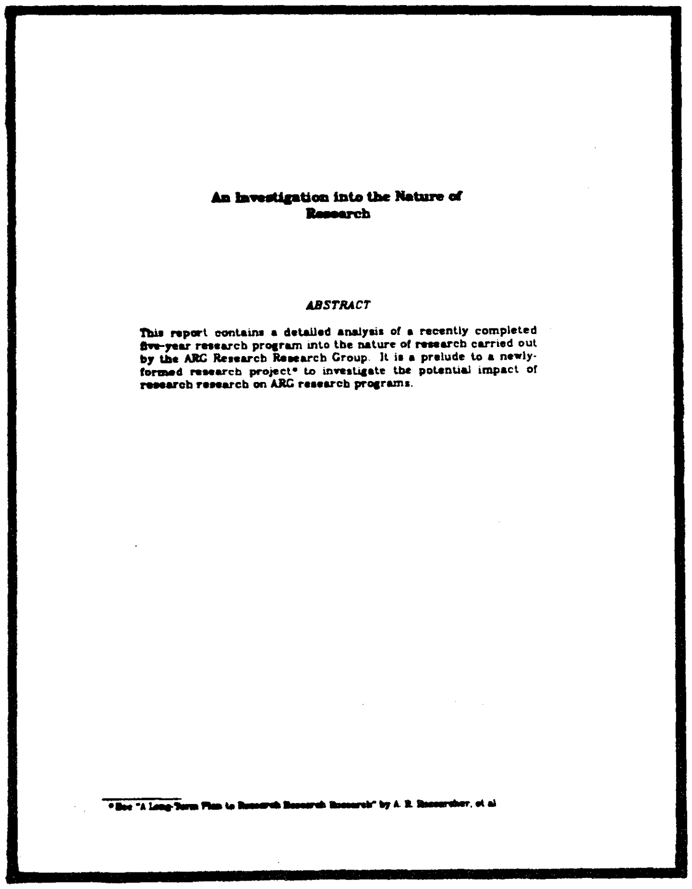
An Investigation into the Nature of Research

A. R. Researcher

Research Research Group
Applied Research Group
Tektronix Labs

February 3, 1981

This information is confidential and no further disclosure thereof can be made to other than Tektronix personnel without written authorization from the Director of Tek Labs, Tektronix, Inc., Beaverton, Oregon.

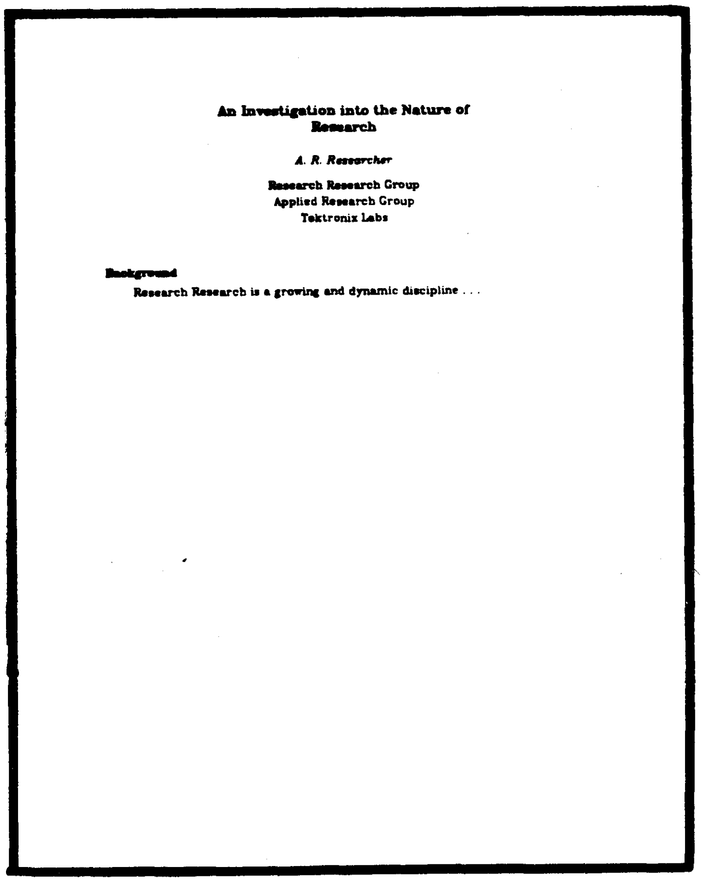


An Investigation into the Nature of Research

ABSTRACT

This report contains a detailed analysis of a recently completed five-year research program into the nature of research carried out by the ARC Research Research Group. It is a prelude to a newly-formed research project* to investigate the potential impact of research research on ARC research programs.

* See "A Long-Term Plan to Research Research Research" by A. R. Researcher, et al



An Investigation into the Nature of Research

A. R. Researcher

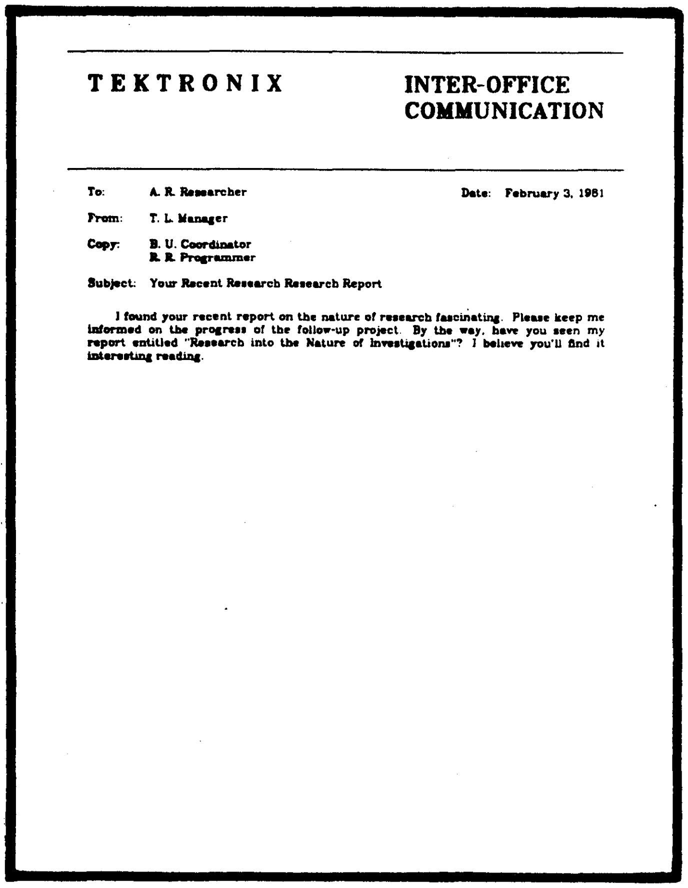
Research Research Group
Applied Research Group
Tektronix Labs

Background

Research Research is a growing and dynamic discipline . . .

Example — An IOC

```
.IOC H
.TO A. R. Researcher
.FR T. L. Manager
.CC B. U. Coordinator
.CC R. R. Programmer
.SU Your Recent Research Research Report
.PP
I found your recent report on the nature of research
fascinating.
Please keep me informed on the progress of the
follow-up project.
By the way, have you seen my report entitled "Research
into the Nature of Investigations"?
I believe you'll find it interesting reading.
```



TEKTRONIX **INTER-OFFICE COMMUNICATION**

To: A. R. Researcher   Date: February 3, 1981

From: T. L. Manager

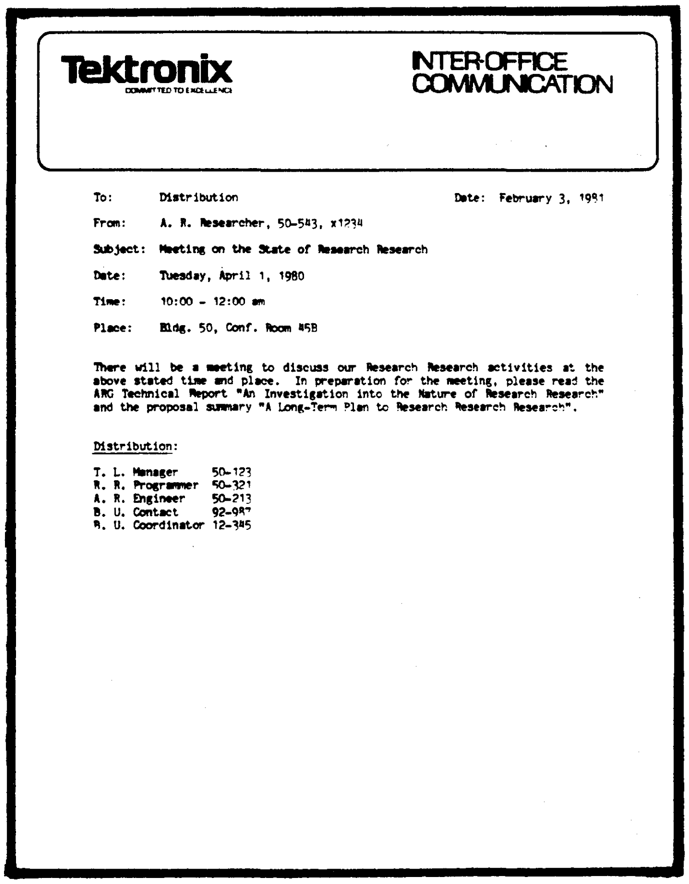
Copy: B. U. Coordinator
R. R. Programmer

Subject: Your Recent Research Research Report

I found your recent report on the nature of research fascinating. Please keep me informed on the progress of the follow-up project. By the way, have you seen my report entitled "Research into the Nature of Investigations"? I believe you'll find it interesting reading.

Example — An IOC Announcing a Meeting

```
.IOC
.TO
T. L. Manager           50-123
R. R. Programmer        50-321
A. R. Engineer          50-213
B. U. Contact           92-987
B. U. Coordinator       12-345
.FR A. R. Researcher, 50-543, x1234
.SU Meeting on the State of Research Research
.DA Tuesday, April 1, 1980
.TI 10:00 - 12:00 am
.PL Bldg. 50, Conf. Room 45B
.LP
There will be a meeting to discuss our Research Research
activities at the above stated time and place.
In preparation for the meeting, please read the ARG
Technical Report "An Investigation into the Nature of
Research Research" and the proposal summary "A Long-Term
Plan to Research Research Research".
```



Tektronix
COMMITTED TO EXCELLENCE

INTER-OFFICE COMMUNICATION

To: Distribution　　　　Date: February 3, 1981

From: A. R. Researcher, 50-543, x1234

Subject: Meeting on the State of Research Research

Date: Tuesday, April 1, 1980

Time: 10:00 - 12:00 am

Place: Bldg. 50, Conf. Room 45B

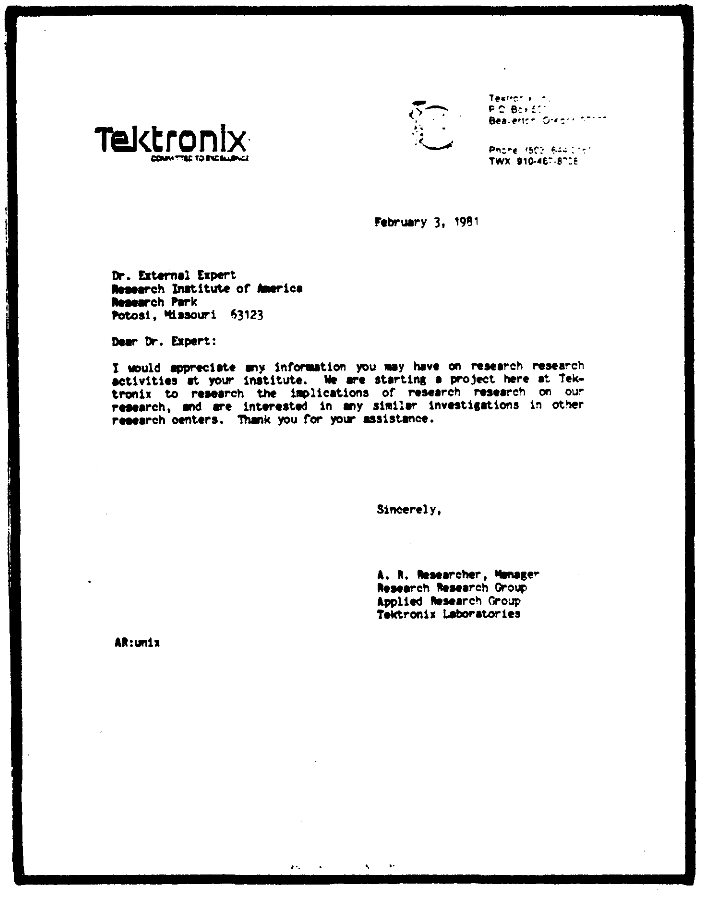
There will be a meeting to discuss our Research Research activities at the above stated time and place. In preparation for the meeting, please read the ARG Technical Report "An Investigation into the Nature of Research Research" and the proposal summary "A Long-Term Plan to Research Research Research".

<u>Distribution:</u>

```
T. L. Manager     50-123
R. R. Programmer  50-321
A. R. Engineer    50-213
B. U. Contact     92-987
B. U. Coordinator 12-345
```

Example — A Business Letter

```
.LT 40
Dr. External Expert
Research Institute of America
Research Park
Potosi, Missouri  63123
.SP
Dear Dr. Expert:
.LP
I would appreciate any information you may have on
research research activities at your institute.
We are starting a project here at Tektronix to research
the implications of research research on our research,
and are interested in any similar investigations in other
research centers.
Thank you for your assistance.
.SP 5
.DS I 40
Sincerely,
.SP 4
A. R. Researcher, Manager
Research Research Group
Applied Research Group
Tektronix Laboratories
.DE
AR:unix
```



Tektronix
COMMITTED TO EXCELLENCE

Tektronix [illegible]
P O Box [illegible]
Beaverton [illegible]

Phone (503) 644-[illegible]
TWX 910-467-[illegible]

February 3, 1981

Dr. External Expert
Research Institute of America
Research Park
Potosi, Missouri 63123

Dear Dr. Expert:

I would appreciate any information you may have on research research activities at your institute. We are starting a project here at Tektronix to research the implications of research research on our research, and are interested in any similar investigations in other research centers. Thank you for your assistance.

Sincerely,

A. R. Researcher, Manager
Research Research Group
Applied Research Group
Tektronix Laboratories

AR:unix

Example — Input Text for Table 5

```
.HL
.DS C
.SP
.LG
.TC "Table 5 \*- Quick Reference:  In-Line Commands"
.NL
.DE
.HL
.SP 1
Note that in-line commands appearing in command and string
register definitions (.de and .ds) should be entered with
\FItwo\FR backslashes (\e\e).
.SP
.DS B
.TA 12n
 \espaceⓉUnpaddable Space Character
 \eeⓉEcho Backslash Character
 \e%ⓉSuppress Hyphenation
 \eF\FIx\FRⓉSwitch to Font \FIx\FR (Also \ef)
 \eF(\FIxy\FRⓉSwitch to Font \FIxy\FR (Also \ef)
 \es\FIn\FRⓉSet Character Size to \FIn\FR Points
 \es\(+-\FIn\FRⓉIncrease/Decrease Size by \FIn\FR Points
 \e(\FIxy\FRⓉSpecial Character \FIxy\FR
 \eo\'...\'ⓉOverstrike Characters
 \e"ⓉIgnore Rest of Input Line (For Comments)
 \e*{ⓉStart Superscript
 \e*}ⓉEnd Superscript
 \e*[ⓉStart Subscript
 \e*]ⓉEnd Subscript
 \e*\FIx\FRⓉIncrement and Print Auto Number \FIx\FR
 \en\FIx\FRⓉPrint Auto Number \FIx\FR (no incr.)
 \e*(DTⓉToday's Date
 \e*(DYⓉToday's Date (Changeable via .ND)
 \e*(DWⓉDay of the Week
 \en(PNⓉCurrent Page Number
.DE
```

Section 6

A TROFF TUTORIAL

INTRODUCTION

troff, a text formatter that drives a typesetter, was developed at Bell Laboratories and is licensed by Western Electric for use on the 8560. The remainder of this section is a reprint of an article describing *troff*. The Technical Notes section of this manual describes the limitations of this program and any changes made to this program by Tektronix.

A TROFF Tutorial

Brian W. Kernighan

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

troff is a text-formatting program for driving the Graphic Systems phototypesetter on the UNIX† and GCOS operating systems. This device is capable of producing high quality text; this paper is an example of **troff** output.

The phototypesetter itself normally runs with four fonts, containing roman, italic and bold letters (as on this page), a full greek alphabet, and a substantial number of special characters and mathematical symbols. Characters can be printed in a range of sizes, and placed anywhere on the page.

troff allows the user full control over fonts, sizes, and character positions, as well as the usual features of a formatter — right-margin justification, automatic hyphenation, page titling and numbering, and so on. It also provides macros, arithmetic variables and operations, and conditional testing, for complicated formatting tasks.

This document is an introduction to the most basic use of **troff**. It presents just enough information to enable the user to do simple formatting tasks like making viewgraphs, and to make incremental changes to existing packages of **troff** commands. In most respects, the UNIX formatter **nroff** is identical to **troff**, so this document also serves as a tutorial on **nroff**.

August 4, 1978

†UNIX is a Trademark of Bell Laboratories.

A TROFF Tutorial

Brian W. Kernighan

Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

troff [1] is a text-formatting program, written by J. F. Ossanna, for producing high-quality printed output from the phototypesetter on the UNIX and GCOS operating systems. This document is an example of **troff** output.

The single most important rule of using **troff** is not to use it directly, but through some intermediary. In many ways, **troff** resembles an assembly language — a remarkably powerful and flexible one — but nonetheless such that many operations must be specified at a level of detail and in a form that is too hard for most people to use effectively.

For two special applications, there are programs that provide an interface to **troff** for the majority of users. **eqn** [2] provides an easy to learn language for typesetting mathematics; the **eqn** user need know no **troff** whatsoever to typeset mathematics. **tbl** [3] provides the same convenience for producing tables of arbitrary complexity.

For producing straight text (which may well contain mathematics or tables), there are a number of 'macro packages' that define formatting rules and operations for specific styles of documents, and reduce the amount of direct contact with **troff**. In particular, the '−ms' [4] and PWB/MM [5] packages for Bell Labs internal memoranda and external papers provide most of the facilities needed for a wide range of document preparation. (This memo was prepared with '−ms'.) There are also packages for viewgraphs, for simulating the older **roff** formatters on UNIX and GCOS, and for other special applications. Typically you will find these packages easier to use than **troff** once you get beyond the most trivial operations; you should always consider them first.

In the few cases where existing packages don't do the whole job, the solution is *not* to write an entirely new set of **troff** instructions from scratch, but to make small changes to adapt packages that already exist.

In accordance with this philosophy of letting someone else do the work, the part of **troff** described here is only a small part of the whole, although it tries to concentrate on the more useful parts. In any case, there is no attempt to be complete. Rather, the emphasis is on showing how to do simple things, and how to make incremental changes to what already exists. The contents of the remaining sections are:

The **troff** described here is the C-language version running on UNIX at Murray Hill, as documented in [1].

To use **troff** you have to prepare not only the actual text you want printed, but some information that tells *how* you want it printed. (Readers who use **roff** will find the approach familiar.) For **troff** the text and the formatting information are often intertwined quite intimately. Most commands to **troff** are placed on a line separate from the text itself, beginning with a period (one command per line). For example,

```
Some text.
.ps 14
Some more text.
```

will change the 'point size', that is, the size of the letters being printed, to '14 point' (one point is 1/72 inch) like this:

Some text. Some more text.

Occasionally, though, something special occurs in the middle of a line — to produce

Area = πr^2

you have to type

```
Area = \(*p\fIr\fR\|\s8\u2\d\s0
```

(which we will explain shortly). The backslash character \ is used to introduce **troff** commands and special characters within a line of text.

2. Point Sizes; Line Spacing

As mentioned above, the command **.ps** sets the point size. One point is 1/72 inch, so 6-point characters are at most 1/12 inch high, and 36-point characters are ½ inch. There are 15 point sizes, listed below.

6 point: Pack my box with five dozen liquor jugs.
7 point: Pack my box with five dozen liquor jugs.
8 point: Pack my box with five dozen liquor jugs.
9 point: Pack my box with five dozen liquor jugs.
10 point: Pack my box with five dozen liquor
11 point: Pack my box with five dozen
12 point: Pack my box with five dozen
14 point: Pack my box with five
16 point 18 point 20 point
22 24 28 36

If the number after .ps is not one of these legal sizes, it is rounded up to the next valid value, with a maximum of 36. If no number follows **.ps**, **troff** reverts to the previous size, whatever it was. **troff** begins with point size 10, which is usually fine. This document is in 9 point.

The point size can also be changed in the middle of a line or even a word with the in-line command **\s**. To produce

UNIX runs on a PDP-11/45

type

```
\s8UNIX\s10 runs on a \s8PDP-\s1011/45
```

As above, **\s** should be followed by a legal point size, except that **\s0** causes the size to revert to its previous value. Notice that **\s1011** can be understood correctly as 'size 10, followed by an 11', if the size is legal, but not otherwise. Be cautious with similar constructions.

Relative size changes are also legal and useful:

```
\s−2UNIX\s+2
```

temporarily decreases the size, whatever it is, by two points, then restores it. Relative size changes have the advantage that the size difference is independent of the starting size of the document. The amount of the relative change is restricted to a single digit.

The other parameter that determines what the type looks like is the spacing between lines, which is set independently of the point size. Vertical spacing is measured from the bottom of one line to the bottom of the next. The command to control vertical spacing is .vs. For running text, it is usually best to set the vertical spacing about 20% bigger than the character size. For example, so far in this document, we have used "9 on 11", that is,

```
.ps 9
.vs 11p
```

If we changed to

```
.ps 9
.vs 9p
```

the running text would look like this. After a few lines, you will agree it looks a little cramped. The right vertical spacing is partly a matter of taste, depending on how much text you want to squeeze into a given space, and partly a matter of traditional printing style. By default, **troff** uses 10 on 12.

Point size and vertical spacing make a substantial difference in the amount of text per square inch. This is 12 on 14.

Point size and vertical spacing make a substantial difference in the amount of text per square inch. For example, 10 on 12 uses about twice as much space as 7 on 8. This is 6 on 7, which is even smaller. It packs a lot more words per line, but you can go blind trying to read it.

When used without arguments, **.ps** and **.vs** revert to the previous size and vertical spacing respectively.

The command **.sp** is used to get extra vertical space. Unadorned, it gives you one extra blank line (one **.vs**, whatever that has been set to). Typically, that's more or less than you want, so **.sp** can be followed by information about how much space you want —

```
.sp 2i
```

means 'two inches of vertical space'.

```
.sp 2p
```

means 'two points of vertical space'; and

```
.sp 2
```

means 'two vertical spaces' — two of whatever

.vs is set to (this can also be made explicit with **.sp 2v**); **troff** also understands decimal fractions in most places, so

```
.sp 1.5i
```

is a space of 1.5 inches. These same scale factors can be used after .vs to define line spacing, and in fact after most commands that deal with physical dimensions.

It should be noted that all size numbers are converted internally to 'machine units', which are 1/432 inch (1/6 point). For most purposes, this is enough resolution that you don't have to worry about the accuracy of the representation. The situation is not quite so good vertically, where resolution is 1/144 inch (1/2 point).

3. Fonts and Special Characters

troff and the typesetter allow four different fonts at any one time. Normally three fonts (Times roman, italic and bold) and one collection of special characters are permanently mounted.

abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ

The greek, mathematical symbols and miscellany of the special font are listed in Appendix A.

troff prints in roman unless told otherwise. To switch into bold, use the .ft command

```
.ft B
```

and for italics,

```
.ft I
```

To return to roman, use .ft **R**; to return to the previous font, whatever it was, use either .ft **P** or just .ft. The 'underline' command

```
.ul
```

causes the next input line to print in italics. .ul can be followed by a count to indicate that more than one line is to be italicized.

Fonts can also be changed within a line or word with the in-line command **\f**:

bold*face* text

is produced by

```
\fBbold\fIface\fR text
```

If you want to do this so the previous font, whatever it was, is left undisturbed, insert extra **\fP** commands, like this:

```
\fBbold\fP\fIface\fP\fR text\fP
```

Because only the immediately previous font is remembered, you have to restore the previous font after each change or you can lose it. The same is true of .ps and .vs when used without an argument.

There are other fonts available besides the standard set, although you can still use only four at any given time. The command .fp tells **troff** what fonts are physically mounted on the typesetter:

```
.fp 3 H
```

says that the Helvetica font is mounted on position 3. (For a complete list of fonts and what they look like, see the **troff** manual.) Appropriate .fp commands should appear at the beginning of your document if you do not use the standard fonts.

It is possible to make a document relatively independent of the actual fonts used to print it by using font numbers instead of names; for example, \f3 and .ft˜3 mean 'whatever font is mounted at position 3', and thus work for any setting. Normal settings are roman font on 1, italic on 2, bold on 3, and special on 4.

There is also a way to get 'synthetic' bold fonts by overstriking letters with a slight offset. Look at the .bd command in [1].

Special characters have four-character names beginning with \(, and they may be inserted anywhere. For example,

¼ + ½ = ¾

is produced by

```
\(14 + \(12 = \(34
```

In particular, greek letters are all of the form \(∗−, where − is an upper or lower case roman letter reminiscent of the greek. Thus to get

$\Sigma(\alpha\times\beta) \rightarrow \infty$

in bare **troff** we have to type

```
\(*S(\(*a\(mu\(*b) \(-> \(if
```

That line is unscrambled as follows:

\(*S	Σ
(	(
\(*a	α
\(mu	×
\(*b	β
)	)
\(->	→
\(if	∞

A complete list of these special names occurs in Appendix A.

In **eqn** [2] the same effect can be achieved with the input

SIGMA (alpha times beta) −> inf

which is less concise, but clearer to the uninitiated.

Notice that each four-character name is a single character as far as **troff** is concerned — the 'translate' command

```
.tr \(mi\(em
```

is perfectly clear, meaning

```
.tr −—
```

that is, to translate − into —.

Some characters are automatically translated into others: grave ` and acute ´ accents (apostrophes) become open and close single quotes '‾'; the combination of "..." is generally preferable to the double quotes "...". Similarly a typed minus sign becomes a hyphen -. To print an explicit − sign, use \-. To get a backslash printed, use \e.

4. Indents and Line Lengths

troff starts with a line length of 6.5 inches, too wide for 8½×11 paper. To reset the line length, use the .ll command, as in

```
.ll 6i
```

As with .sp, the actual length can be specified in several ways; inches are probably the most intuitive.

The maximum line length provided by the typesetter is 7.5 inches, by the way. To use the full width, you will have to reset the default physical left margin ("page offset"), which is normally slightly less than one inch from the left edge of the paper. This is done by the .po command.

```
.po 0
```

sets the offset as far to the left as it will go.

The indent command .in causes the left margin to be indented by some specified amount from the page offset. If we use .in to move the left margin in, and .ll to move the right margin to the left, we can make offset blocks of text:

```
.in 0.3i
.ll −0.3i
text to be set into a block
.ll +0.3i
.in −0.3i
```

will create a block that looks like this:

> Pater noster qui est in caelis sanctificetur nomen tuum; adveniat regnum tuum; fiat voluntas tua, sicut in caelo, et in terra. ... Amen.

Notice the use of '+' and '−' to specify the amount of change. These change the previous setting by the specified amount, rather than just overriding it. The distinction is quite important: .ll +1i makes lines one inch longer; .ll 1i makes them one inch *long*.

With .in, .ll and .po, the previous value is used if no argument is specified.

To indent a single line, use the 'temporary indent' command .ti. For example, all paragraphs in this memo effectively begin with the command

```
.ti 3
```

Three of what? The default unit for .ti, as for most horizontally oriented commands (.ll, .in, .po), is ems; an em is roughly the width of the letter 'm' in the current point size. (Precisely, a em in size *p* is *p* points.) Although inches are usually clearer than ems to people who don't set type for a living, ems have a place: they are a measure of size that is proportional to the current point size. If you want to make text that keeps its proportions regardless of point size, you should use ems for all dimensions. Ems can be specified as scale factors directly, as in .ti 2.5m.

Lines can also be indented negatively if the indent is already positive:

```
.ti −0.3i
```

causes the next line to be moved back three tenths of an inch. Thus to make a decorative initial capital, we indent the whole paragraph, then move the letter 'P' back with a .ti command:

> Pater noster qui est in caelis sanctificetur nomen tuum; adveniat regnum tuum; fiat voluntas tua, sicut in caelo, et in terra. ... Amen.

Of course, there is also some trickery to make the 'P' bigger (just a '\s36P\s0'), and to move it down from its normal position (see the section on local motions).

5. Tabs

Tabs (the ASCII 'horizontal tab' character) can be used to produce output in columns, or to set the horizontal position of output. Typically tabs are used only in unfilled text. Tab stops are set by default every half inch from the current indent, but can be changed by the .ta command. To set stops every inch, for example,

```
.ta 1i 2i 3i 4i 5i 6i
```

Unfortunately the stops are left-justified only (as on a typewriter), so lining up columns of right-justified numbers can be painful. If you have many numbers, or if you need more complicated table layout, *don't* use **troff** directly; use the **tbl** program described in [3].

For a handful of numeric columns, you can do it this way: Precede every number by enough blanks to make it line up when typed.

```
.nf
.ta 1i 2i 3i
  1 tab   2 tab   3
 40 tab  50 tab  60
700 tab 800 tab 900
.fi
```

Then change each leading blank into the string **\0**. This is a character that does not print, but that has the same width as a digit. When printed, this will produce

```
  1          2          3
 40         50         60
700        800        900
```

It is also possible to fill up tabbed-over space with some character other than blanks by setting the 'tab replacement character' with the **.tc** command:

```
.ta 1.5i 2.5i
.tc \(ru      (\(ru is "_")
Name tab Age tab
```

produces

```
Name ________________ Age __________
```

To reset the tab replacement character to a blank, use **.tc** with no argument. (Lines can also be drawn with the **\l** command, described in Section 6.)

troff also provides a very general mechanism called 'fields' for setting up complicated columns. (This is used by **tbl**). We will not go into it in this paper.

6. Local Motions: Drawing lines and characters

Remember 'Area = πr^2' and the big 'P' in the Paternoster. How are they done? **troff** provides a host of commands for placing characters of any size at any place. You can use them to draw special characters or to tune your output for a particular appearance. Most of these commands are straightforward, but messy to read and tough to type correctly.

If you won't use **eqn**, subscripts and superscripts are most easily done with the half-line local motions **\u** and **\d**. To go back up the page half a point-size, insert a **\u** at the desired place; to go down, insert a **\d**. (**\u** and **\d** should always be used in pairs, as explained below.) Thus

```
Area = \(*pr\u2\d
```

produces

Area = πr^2

To make the '2' smaller, bracket it with **\s−2...\s0**. Since **\u** and **\d** refer to the current point size, be sure to put them either both inside or both outside the size changes, or you will get an unbalanced vertical motion.

Sometimes the space given by **\u** and **\d** isn't the right amount. The **\v** command can be used to request an arbitrary amount of vertical motion. The in-line command

```
\v'(amount)'
```

causes motion up or down the page by the amount specified in '(amount)'. For example, to move the 'P' down, we used

```
.in +0.6i          (move paragraph in)
.ll −0.3i          (shorten lines)
.ti −0.3i          (move P back)
\v'2'\s36P\s0\v'−2'ater noster qui est
in caelis ...
```

A minus sign causes upward motion, while no sign or a plus sign means down the page. Thus **\v'−2'** causes an upward vertical motion of two line spaces.

There are many other ways to specify the amount of motion —

```
\v'0.1i'
\v'3p'
\v'−0.5m'
```

and so on are all legal. Notice that the scale specifier **i** or **p** or **m** goes inside the quotes. Any character can be used in place of the quotes; this is also true of all other **troff** commands described in this section.

Since **troff** does not take within-the-line vertical motions into account when figuring out where it is on the page, output lines can have unexpected positions if the left and right ends aren't at the same vertical position. Thus **\v**, like **\u** and **\d**, should always balance upward vertical motion in a line with the same amount in the downward direction.

Arbitrary horizontal motions are also available — **\h** is quite analogous to **\v**, except that the default scale factor is ems instead of line spaces. As an example,

```
\h'−0.1i'
```

causes a backwards motion of a tenth of an inch. As a practical matter, consider printing the mathematical symbol '>>'. The default spacing is too wide, so **eqn** replaces this by

>\h'−0.3m'>

to produce >>.

Frequently \h is used with the 'width function' \w to generate motions equal to the width of some character string. The construction

\w'thing'

is a number equal to the width of 'thing' in machine units (1/432 inch). All **troff** computations are ultimately done in these units. To move horizontally the width of an 'x', we can say

\h'\w'x'u'

As we mentioned above, the default scale factor for all horizontal dimensions is **m**, ems, so here we must have the **u** for machine units, or the motion produced will be far too large. **troff** is quite happy with the nested quotes, by the way, so long as you don't leave any out.

As a live example of this kind of construction, all of the command names in the text, like .sp, were done by overstriking with a slight offset. The commands for .sp are

.sp\h'−\w'.sp'u'\h'1u'.sp

That is, put out '.sp', move left by the width of '.sp', move right 1 unit, and print '.sp' again. (Of course there is a way to avoid typing that much input for each command name, which we will discuss in Section 11.)

There are also several special-purpose **troff** commands for local motion. We have already seen \0, which is an unpaddable white space of the same width as a digit. 'Unpaddable' means that it will never be widened or split across a line by line justification and filling. There is also \(blank), which is an unpaddable character the width of a space, \|, which is half that width, \^, which is one quarter of the width of a space, and \&, which has zero width. (This last one is useful, for example, in entering a text line which would otherwise begin with a '.'.)

The command \o, used like

\o'set of characters'

causes (up to 9) characters to be overstruck, centered on the widest. This is nice for accents, as in

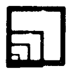
syst\o"e\(ga"me t\o"e\(aa"l\o"e\(aa"phonique

which makes

système téléphonique

The accents are \(ga and \(aa, or \\` and \\'; remember that each is just one character to **troff**.

You can make your own overstrikes with another special convention, \z, the zero-motion command. \zx suppresses the normal horizontal motion after printing the single character **x**, so another character can be laid on top of it. Although sizes can be changed within \o, it centers the characters on the widest, and there can be no horizontal or vertical motions, so \z may be the only way to get what you want:



is produced by

.sp 2
\s8\z\(sq\s14\z\(sq\s22\z\(sq\s36\(sq

The .sp is needed to leave room for the result.

As another example, an extra-heavy semicolon that looks like

; instead of ; or ;

can be constructed with a big comma and a big period above it:

\s+6\z,\v'−0.25m'.\v'0.25m'\s0

'0.25m' is an empirical constant.

A more ornate overstrike is given by the bracketing function \b, which piles up characters vertically, centered on the current baseline. Thus we can get big brackets, constructing them with piled-up smaller pieces:

$$\left\{\left[x \right]\right\}$$

by typing in only this:

.sp
\b'\(lt\(lk\(lb' \b'\(lc\(lf' x \b'\(rc\(rf' \b'\(rt\(rk\(rb'

troff also provides a convenient facility for drawing horizontal and vertical lines of arbitrary length with arbitrary characters. \l'1i' draws a line one inch long, like this: ________________. The length can be followed by the character to use if the _ isn't appropriate; \l'0.5i.' draws a half-inch line of dots: .... The construction \L is entirely analogous, except that it draws a vertical line instead of horizontal.

7. Strings

Obviously if a paper contains a large number of occurrences of an acute accent over a letter 'e', typing \o"e\'" for each é would be a great nuisance.

Fortunately, **troff** provides a way in which you can store an arbitrary collection of text in a 'string', and thereafter use the string name as a shorthand for its contents. Strings are one of several **troff** mechanisms whose judicious use lets you type a document with less effort and organize it so that extensive format changes can be made with few editing changes.

A reference to a string is replaced by whatever text the string was defined as. Strings are defined with the command **.ds**. The line

```
.ds e \o"e\'"
```

defines the string **e** to have the value **\o"e\'"**

String names may be either one or two characters long, and are referred to by ***x** for one character names or ***(xy** for two character names. Thus to get téléphone, given the definition of the string **e** as above, we can say t*el*ephone.

If a string must begin with blanks, define it as

```
.ds xx "      text
```

The double quote signals the beginning of the definition. There is no trailing quote; the end of the line terminates the string.

A string may actually be several lines long; if **troff** encounters a \ at the end of *any* line, it is thrown away and the next line added to the current one. So you can make a long string simply by ending each line but the last with a backslash:

```
.ds xx this \
is a very \
long string
```

Strings may be defined in terms of other strings, or even in terms of themselves; we will discuss some of these possibilities later.

8. Introduction to Macros

Before we can go much further in **troff**, we need to learn a bit about the macro facility. In its simplest form, a macro is just a shorthand notation quite similar to a string. Suppose we want every paragraph to start in exactly the same way — with a space and a temporary indent of two ems:

```
.sp
.ti +2m
```

Then to save typing, we would like to collapse these into one shorthand line, a **troff** 'command' like

```
.PP
```

that would be treated by **troff** exactly as

```
.sp
.ti +2m
```

.PP is called a *macro*. The way we tell **troff** what **.PP** means is to *define* it with the **.de** command:

```
.de PP
.sp
.ti +2m
..
```

The first line names the macro (we used '**.PP**' for 'paragraph', and upper case so it wouldn't conflict with any name that **troff** might already know about). The last line .. marks the end of the definition. In between is the text, which is simply inserted whenever **troff** sees the 'command' or macro call

```
.PP
```

A macro can contain any mixture of text and formatting commands.

The definition of **.PP** has to precede its first use; undefined macros are simply ignored. Names are restricted to one or two characters.

Using macros for commonly occurring sequences of commands is critically important. Not only does it save typing, but it makes later changes much easier. Suppose we decide that the paragraph indent is too small, the vertical space is much too big, and roman font should be forced. Instead of changing the whole document, we need only change the definition of **.PP** to something like

```
.de PP        \" paragraph macro
.sp 2p
.ti +3m
.ft R
..
```

and the change takes effect everywhere we used **.PP**.

\" is a **troff** command that causes the rest of the line to be ignored. We use it here to add comments to the macro definition (a wise idea once definitions get complicated).

As another example of macros, consider these two which start and end a block of offset, unfilled text, like most of the examples in this paper:

```
.de BS        \" start indented block
.sp
.nf
.in +0.3i
..
.de BE        \" end indented block
.sp
.fi
.in −0.3i
..
```

Now we can surround text like

```
Copy to
John Doe
Richard Roberts
Stanley Smith
```

by the commands **.BS** and **.BE**, and it will come out as it did above. Notice that we indented by .in +0.3i instead of .in 0.3i. This way we can nest our uses of **.BS** and **BE** to get blocks within blocks.

If later on we decide that the indent should be 0.5i, then it is only necessary to change the definitions of **.BS** and **.BE**, not the whole paper.

9. Titles, Pages and Numbering

This is an area where things get tougher, because nothing is done for you automatically. Of necessity, some of this section is a cookbook, to be copied literally until you get some experience.

Suppose you want a title at the top of each page, saying just

```
~~~~left top          center top          right top~~~~
```

In **roff**, one can say

```
.he 'left top'center top'right top'
.fo 'left bottom'center bottom'right bottom'
```

to get headers and footers automatically on every page. Alas, this doesn't work in **troff**, a serious hardship for the novice. Instead you have to do a lot of specification.

You have to say what the actual title is (easy); when to print it (easy enough); and what to do at and around the title line (harder). Taking these in reverse order, first we define a macro .NP (for 'new page') to process titles and the like at the end of one page and the beginning of the next:

```
.de NP
'bp
'sp 0.5i
.tl 'left top'center top'right top'
'sp 0.3i
..
```

To make sure we're at the top of a page, we issue a 'begin page' command **'bp**, which causes a skip to top-of-page (we'll explain the ' shortly). Then we space down half an inch, print the title (the use of **.tl** should be self explanatory; later we will discuss parameterizing the titles), space another 0.3 inches, and we're done.

To ask for .**NP** at the bottom of each page, we have to say something like 'when the text is within an inch of the bottom of the page, start the processing for a new page.' This is done with a 'when' command **.wh**:

```
.wh  −1i  NP
```

(No '.' is used before NP; this is simply the name of a macro, not a macro call.) The minus sign means 'measure up from the bottom of the page', so '−1i' means 'one inch from the bottom'.

The **.wh** command appears in the input outside the definition of **.NP**; typically the input would be

```
.de NP
...
..
.wh −1i NP
```

Now what happens? As text is actually being output, **troff** keeps track of its vertical position on the page, and after a line is printed within one inch from the bottom, the **.NP** macro is activated. (In the jargon, the .wh command sets a *trap* at the specified place, which is 'sprung' when that point is passed.) **.NP** causes a skip to the top of the next page (that's what the **'bp** was for), then prints the title with the appropriate margins.

Why **'bp** and **'sp** instead of **.bp** and **.sp**? The answer is that **.sp** and **.bp**, like several other commands, cause a *break* to take place. That is, all the input text collected but not yet printed is flushed out as soon as possible, and the next input line is guaranteed to start a new line of output. If we had used **.sp** or **.bp** in the **.NP** macro, this would cause a break in the middle of the current output line when a new page is started. The effect would be to print the left-over part of that line at the top of the page, followed by the next input line on a new output line. This is *not* what we want. Using ' instead of . for a command tells **troff** that no break is to take place — the output line currently being filled should *not* be forced out before the space or new page.

The list of commands that cause a break is short and natural:

```
.bp  .br  .ce  .fi  .nf  .sp  .in  .ti
```

All others cause *no* break, regardless of whether you use a . or a '. If you really need a break, add a **.br** command at the appropriate place.

One other thing to beware of — if you're changing fonts or point sizes a lot, you may find that if you cross a page boundary in an unexpected font or size, your titles come out in that size and font instead of what you intended. Furthermore, the length of a title is independent of the current line length, so titles will come out at the default length of 6.5 inches unless you change it, which is done with the .lt command.

There are several ways to fix the problems of point sizes and fonts in titles. For the simplest applications, we can change **.NP** to set the proper size and font for the title, then restore the previous values, like this:

```
.de NP
'bp
'sp 0.5i
.ft R           \" set title font to roman
.ps 10          \" and size to 10 point
.lt 6i          \" and length to 6 inches
.tl 'left'center'right'
.ps             \" revert to previous size
.ft P           \" and to previous font
'sp 0.3i
..
```

This version of **.NP** does *not* work if the fields in the .tl command contain size or font changes. To cope with that requires **troff**'s 'environment' mechanism, which we will discuss in Section 13.

To get a footer at the bottom of a page, you can modify **.NP** so it does some processing before the **'bp** command, or split the job into a footer macro invoked at the bottom margin and a header macro invoked at the top of the page. These variations are left as exercises.

Output page numbers are computed automatically as each page is produced (starting at 1), but no numbers are printed unless you ask for them explicitly. To get page numbers printed, include the character **%** in the .tl line at the position where you want the number to appear. For example

```
.tl ''- % -''
```

centers the page number inside hyphens, as on this page. You can set the page number at any time with either **.bp n**, which immediately starts a new page numbered **n**, or with **.pn n**, which sets the page number for the next page but doesn't cause a skip to the new page. Again, **.bp +n** sets the page number to **n** more than its current value; **.bp** means **.bp +**1.

10. Number Registers and Arithmetic

troff has a facility for doing arithmetic, and for defining and using variables with numeric values, called *number registers.* Number registers, like strings and macros, can be useful in setting up a document so it is easy to change later. And of course they serve for any sort of arithmetic computation.

Like strings, number registers have one or two character names. They are set by the **.nr** command, and are referenced anywhere by **\nx** (one character name) or **\n(xy** (two character name).

There are quite a few pre-defined number registers maintained by **troff**, among them **%** for the current page number; **nl** for the current vertical position on the page; **dy**, **mo** and **yr** for the current day, month and year; and **.s** and **.f** for the current size and font. (The font is a number from 1 to 4.) Any of these can be used in computations like any other register, but some, like .s and .f, cannot be changed with **.nr**.

As an example of the use of number registers, in the **−ms** macro package [4], most significant parameters are defined in terms of the values of a handful of number registers. These include the point size for text, the vertical spacing, and the line and title lengths. To set the point size and vertical spacing for the following paragraphs, for example, a user may say

```
.nr PS 9
.nr VS 11
```

The paragraph macro **.PP** is defined (roughly) as follows:

```
.de PP
.ps \\n(PS          \" reset size
.vs \\n(VSp         \" spacing
.ft R               \" font
.sp 0.5v            \" half a line
.ti +3m
..
```

This sets the font to Roman and the point size and line spacing to whatever values are stored in the number registers **PS** and **VS**.

Why are there two backslashes? This is the eternal problem of how to quote a quote. When **troff** originally reads the macro definition, it peels off one backslash to see what's coming next. To ensure that another is left in the definition when the macro is *used,* we have to put in two backslashes in the definition. If only one backslash is used, point size and vertical spacing will be frozen at the time the macro is defined, not when it is used.

Protecting by an extra layer of backslashes is only needed for \n, *, \$ (which we haven't come to yet), and \ itself. Things like \s, \f, \h, \v, and so on do not need an extra backslash, since they are converted by **troff** to an internal code immediately upon being seen.

Arithmetic expressions can appear anywhere that a number is expected. As a trivial example,

```
.nr PS \\n(PS−2
```

decrements PS by 2. Expressions can use the arithmetic operators +, −, *, /, % (mod), the relational operators >, >=, <, <=, =, and != (not equal), and parentheses.

Although the arithmetic we have done so far has been straightforward, more complicated things are somewhat tricky. First, number registers hold only integers. **troff** arithmetic uses truncating integer division, just like Fortran. Second, in the absence of parentheses, evaluation is done left-to-right without any operator precedence (including relational operators). Thus

```
7*−4+3/13
```

becomes '−1'. Number registers can occur anywhere in an expression, and so can scale indicators like **p**, **i**, **m**, and so on (but no spaces). Although integer division causes truncation, each number and its scale indicator is converted to machine units (1/432 inch) before any arithmetic is done, so 1i/2u evaluates to 0.5i correctly.

The scale indicator **u** often has to appear when you wouldn't expect it — in particular, when arithmetic is being done in a context that implies horizontal or vertical dimensions. For example,

```
.ll 7/2i
```

would seem obvious enough — 3½ inches. Sorry. Remember that the default units for horizontal parameters like **.ll** are ems. That's really '7 ems / 2 inches', and when translated into machine units, it becomes zero. How about

```
.ll 7i/2
```

Sorry, still no good — the '2' is '2 ems', so '7i/2' is small, although not zero. You *must* use

```
.ll 7i/2u
```

So again, a safe rule is to attach a scale indicator to every number, even constants.

For arithmetic done within a **.nr** command, there is no implication of horizontal or vertical dimension, so the default units are 'units', and 7i/2 and 7i/2u mean the same thing. Thus

```
.nr ll 7i/2
.ll \\n(llu
```

does just what you want, so long as you don't forget the **u** on the **.ll** command.

11. Macros with arguments

The next step is to define macros that can change from one use to the next according to parameters supplied as arguments. To make this work, we need two things: first, when we define the macro, we have to indicate that some parts of it will be provided as arguments when the macro is called. Then when the macro is called we have to provide actual arguments to be plugged into the definition.

Let us illustrate by defining a macro **.SM** that will print its argument two points smaller than the surrounding text. That is, the macro call

```
.SM TROFF
```

will produce TROFF.

The definition of **.SM** is

```
.de SM
\s−2\\$1\s+2
..
```

Within a macro definition, the symbol **\\$n** refers to the **n**th argument that the macro was called with. Thus **\\$1** is the string to be placed in a smaller point size when **.SM** is called.

As a slightly more complicated version, the following definition of **.SM** permits optional second and third arguments that will be printed in the normal size:

```
.de SM
\\$3\s−2\\$1\s+2\\$2
..
```

Arguments not provided when the macro is called are treated as empty, so

```
.SM TROFF ),
```

produces TROFF), while

```
.SM TROFF ). (
```

produces (TROFF). It is convenient to reverse the order of arguments because trailing punctuation is much more common than leading.

By the way, the number of arguments that a macro was called with is available in number register **.$**.

The following macro **.BD** is the one used to make the 'bold roman' we have been using for **troff** command names in text. It combines horizontal motions, width computations, and argument rearrangement.

```
.de BD
\&\\$3\f1\\$1\h'−\w'\\$1'u+1u'\\$1\fP\\$2
..
```

The **\h** and **\w** commands need no extra backslash, as we discussed above. The **\&** is there in case the argument begins with a period.

Two backslashes are needed with the **\\$n** commands, though, to protect one of them when the macro is being defined. Perhaps a second example will make this clearer. Consider a macro called .SH which produces section headings rather like those in this paper, with the sections numbered automatically, and the title in bold in a smaller size. The use is

```
.SH  "Section title ..."
```

(If the argument to a macro is to contain blanks, then it must be *surrounded* by double quotes, unlike a string, where only one leading quote is permitted.)

Here is the definition of the .SH macro:

```
.nr SH 0        \" initialize section number
.de SH
.sp 0.3i
.ft B
.nr SH \\n(SH+1   \" increment number
.ps \\n(PS−1      \" decrease PS
\\n(SH. \\$1      \" number. title
.ps \\n(PS        \" restore PS
.sp 0.3i
.ft R
..
```

The section number is kept in number register SH, which is incremented each time just before it is used. (A number register may have the same name as a macro without conflict but a string may not.)

We used **\\n(SH** instead of **\n(SH** and **\\n(PS** instead of **\n(PS**. If we had used **\n(SH**, we would get the value of the register at the time the macro was *defined,* not at the time it was *used.* If that's what you want, fine, but not here. Similarly, by using **\\n(PS**, we get the point size at the time the macro is called.

As an example that does not involve numbers, recall our **.NP** macro which had a

```
.tl 'left'center'right'
```

We could make these into parameters by using instead

```
.tl '\\*(LT'\\*(CT'\\*(RT'
```

so the title comes from three strings called LT, CT and RT. If these are empty, then the title will be a blank line. Normally CT would be set with something like

```
.ds CT - % -
```

to give just the page number between hyphens (as on the top of this page), but a user could supply private definitions for any of the strings.

12. Conditionals

Suppose we want the .SH macro to leave two extra inches of space just before section 1, but nowhere else. The cleanest way to do that is to test inside the .SH macro whether the section number is 1, and add some space if it is. The .if command provides the conditional test that we can add just before the heading line is output:

```
.if \\n(SH=1 .sp 2i        \" first section only
```

The condition after the .if can be any arithmetic or logical expression. If the condition is logically true, or arithmetically greater than zero, the rest of the line is treated as if it were text — here a command. If the condition is false, or zero or negative, the rest of the line is skipped.

It is possible to do more than one command if a condition is true. Suppose several operations are to be done before section 1. One possibility is to define a macro .S1 and invoke it if we are about to do section 1 (as determined by an .if).

```
.de S1
--- processing for section 1 ---
..
.de SH
...
.if \\n(SH=1 .S1
...
..
```

An alternate way is to use the extended form of the .if, like this:

```
.if \\n(SH=1 \{--- processing
for section 1 ----\}
```

The braces \{ and \} must occur in the positions shown or you will get unexpected extra lines in your output. **troff** also provides an 'if-else' construction, which we will not go into here.

A condition can be negated by preceding it with !; we get the same effect as above (but less clearly) by using

```
.if !\\n(SH>1 .S1
```

There are a handful of other conditions that can be tested with .if. For example, is the current page even or odd?

```
.if e .tl ''even page title''
.if o .tl ''odd page title''
```

gives facing pages different titles when used inside an appropriate new page macro.

Two other conditions are t and n, which tell you whether the formatter is **troff** or **nroff**.

```
.if t troff stuff ...
.if n nroff stuff ...
```

Finally, string comparisons may be made in an .if:

```
.if 'string1'string2' stuff
```

does 'stuff' if *string1* is the same as *string2*. The character separating the strings can be anything reasonable that is not contained in either string. The strings themselves can reference strings with \*, arguments with **\\$**, and so on.

13. Environments

As we mentioned, there is a potential problem when going across a page boundary: parameters like size and font for a page title may well be different from those in effect in the text when the page boundary occurs. **troff** provides a very general way to deal with this and similar situations. There are three 'environments', each of which has independently settable versions of many of the parameters associated with processing, including size, font, line and title lengths, fill/nofill mode, tab stops, and even partially collected lines. Thus the titling problem may be readily solved by processing the main text in one environment and titles in a separate one with its own suitable parameters.

The command **.ev n** shifts to environment **n**; **n** must be 0, 1 or 2. The command **.ev** with no argument returns to the previous environment. Environment names are maintained in a stack, so calls for different environments may be nested and unwound consistently.

Suppose we say that the main text is processed in environment 0, which is where **troff** begins by default. Then we can modify the new page macro **.NP** to process titles in environment 1 like this:

```
.de NP
.ev 1         \" shift to new environment
.lt 6i        \" set parameters here
.ft R
.ps 10
... any other processing ...
.ev           \" return to previous environment
..
```

It is also possible to initialize the parameters for an environment outside the .NP macro, but the version shown keeps all the processing in one place and is thus easier to understand and change.

14. Diversions

There are numerous occasions in page layout when it is necessary to store some text for a period of time without actually printing it. Footnotes are the most obvious example: the text of the footnote usually appears in the input well before the place on the page where it is to be printed is reached. In fact, the place where it is output normally depends on how big it is, which implies that there must be a way to process the footnote at least enough to decide its size without printing it.

troff provides a mechanism called a diversion for doing this processing. Any part of the output may be diverted into a macro instead of being printed, and then at some convenient time the macro may be put back into the input.

The command **.di xy** begins a diversion — all subsequent output is collected into the macro **xy** until the command **.di** with no arguments is encountered. This terminates the diversion. The processed text is available at any time thereafter, simply by giving the command

```
.xy
```

The vertical size of the last finished diversion is contained in the built-in number register **dn**.

As a simple example, suppose we want to implement a 'keep-release' operation, so that text between the commands **.KS** and **.KE** will not be split across a page boundary (as for a figure or table). Clearly, when a **.KS** is encountered, we have to begin diverting the output so we can find out how big it is. Then when a **.KE** is seen, we decide whether the diverted text will fit on the current page, and print it either there if it fits, or at the top of the next page if it doesn't. So:

```
.de KS      \" start keep
.br         \" start fresh line
.ev 1       \" collect in new environment
.fi         \" make it filled text
.di XX      \" collect in XX
..

.de KE      \" end keep
.br         \" get last partial line
.di         \" end diversion
.if \\n(dn>=\\n(.t .bp  \" bp if doesn't fit
.nf         \" bring it back in no-fill
.XX         \" text
.ev         \" return to normal environment
..
```

Recall that number register **nl** is the current position on the output page. Since output was being diverted, this remains at its value when the diversion started. **dn** is the amount of text in the diversion; .t (another built-in register) is the distance to the next trap, which we assume is at the bottom margin of the page. If the diversion is large enough to go past the trap, the **.if** is satisfied, and a **.bp** is issued. In either case, the diverted output is then brought back with **.XX**. It is essential to bring it back in no-fill mode so **troff** will do no further processing on it.

This is not the most general keep-release, nor is it robust in the face of all conceivable inputs, but it would require more space than we have here to write it in full generality. This section is not intended to teach everything about diversions, but to sketch out enough that you can read existing macro packages with some comprehension.

Acknowledgements

I am deeply indebted to J. F. Ossanna, the author of **troff**, for his repeated patient explanations of fine points, and for his continuing willingness to adapt **troff** to make other uses easier. I am also grateful to Jim Blinn, Ted Dolotta, Doug McIlroy, Mike Lesk and Joel Sturman for helpful comments on this paper.

References

[1] J. F. Ossanna, *NROFF/TROFF* User's Manual, Bell Laboratories Computing Science Technical Report 54, 1976.

[2] B. W. Kernighan, *A System for Typesetting Mathematics — User's Guide (Second Edition),* Bell Laboratories Computing Science Technical Report 17, 1977.

[3] M. E. Lesk, *TBL — A Program to Format Tables,* Bell Laboratories Computing Science Technical Report 49, 1976.

[4] M. E. Lesk, *Typing Documents on UNIX,* Bell Laboratories, 1978.

[5] J. R. Mashey and D. W. Smith, *PWB/MM — Programmer's Workbench Memorandum Macros,* Bell Laboratories internal memorandum.

Appendix A: Phototypesetter Character Set

These characters exist in roman, italic, and bold. To get the one on the left, type the four-character name on the right.

ff	\(ff	fi	\(fi	fl	\(fl	ffi	\(Fi	ffl	\(Fl
_	\(ru	—	\(em	¼	\(14	½	\(12	¾	\(34
©	\(co	°	\(de	†	\(dg	′	\(fm	¢	\(ct
®	\(rg	●	\(bu	□	\(sq	-	\(hy		

(In bold, \(sq is ■.)

The following are special-font characters:

+	\(pl	−	\(mi	×	\(mu	÷	\(di
=	\(eq	≡	\(==	≥	\(>=	≤	\(<=
≠	\(!=	±	\(+-	¬	\(no	/	\(sl
~	\(ap	≃	\(~=	∝	\(pt	∇	\(gr
→	\(->	←	\(<-	↑	\(ua	↓	\(da
∫	\(is	∂	\(pd	∞	\(if	√	\(sr
⊂	\(sb	⊃	\(sp	∪	\(cu	∩	\(ca
⊆	\(ib	⊇	\(ip	∈	\(mo	∅	\(es
´	\(aa	`	\(ga	○	\(ci	Ⓑ	\(bs
§	\(sc	‡	\(dd	☚	\(lh	☛	\(rh
⎧	\(lt	⎫	\(rt	⌈	\(lc	⌉	\(rc
⎩	\(lb	⎭	\(rb	⌊	\(lf	⌋	\(rf
⎨	\(lk	⎬	\(rk	│	\(bv	ς	\(ts
│	\(br	|	\(or	_	\(ul	‾	\(rn
∗	\(**						

These four characters also have two-character names. The ´ is the apostrophe on terminals; the ` is the other quote mark.

´ \´   ` \`   − \-   _ _

These characters exist only on the special font, but they do not have four-character names:

" { } < > ~ ^ \ # @

For greek, precede the roman letter by \(∗ to get the corresponding greek; for example, \(∗a is α.

a b g d e z y h i k l m n c o p r s t u f x q w
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω

A B G D E Z Y H I K L M N C O P R S T U F X Q W
Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω

Section 7

EQN—A SYSTEM FOR TYPESETTING MATHEMATICS

INTRODUCTION

eqn, a system for typesetting mathematics, was developed at Bell Laboratories and is licensed by Western Electric for use on the 8560. The remainder of this section is a reprint of an article describing *eqn*. The Technical Notes section of this manual describes the limitations of this program and any changes made to this program by Tektronix.

A System for Typesetting Mathematics

Brian W. Kernighan and Lorinda L. Cherry

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This paper describes the design and implementation of a system for typesetting mathematics. The language has been designed to be easy to learn and to use by people (for example, secretaries and mathematical typists) who know neither mathematics nor typesetting. Experience indicates that the language can be learned in an hour or so, for it has few rules and fewer exceptions. For typical expressions, the size and font changes, positioning, line drawing, and the like necessary to print according to mathematical conventions are all done automatically. For example, the input

sum from i=0 to infinity x sub i = pi over 2

produces

$$\sum_{i=0}^{\infty} x_i = \frac{\pi}{2}$$

The syntax of the language is specified by a small context-free grammar; a compiler-compiler is used to make a compiler that translates this language into typesetting commands. Output may be produced on either a phototypesetter or on a terminal with forward and reverse half-line motions. The system interfaces directly with text formatting programs, so mixtures of text and mathematics may be handled simply.

This paper is a revision of a paper originally published in CACM, March, 1975.

1. Introduction

"Mathematics is known in the trade as *difficult*, or *penalty, copy* because it is slower, more difficult, and more expensive to set in type than any other kind of copy normally occurring in books and journals." [1]

One difficulty with mathematical text is the multiplicity of characters, sizes, and fonts. An expression such as

$$\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$$

requires an intimate mixture of roman, italic and greek letters, in three sizes, and a special character or two. ("Requires" is perhaps the wrong word, but mathematics has its own typographical conventions which are quite different from those of ordinary text.) Typesetting such an expression by traditional methods is still an essentially manual operation.

A second difficulty is the two dimensional character of mathematics, which the superscript and limits in the preceding example showed in its simplest form. This is carried further by

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \cdots}}}$$

and still further by

$$\int \frac{dx}{ae^{mx} - be^{-mx}} = \left\{ \begin{array}{ll} \frac{1}{2m\sqrt{ab}} & \log \frac{\sqrt{a}\,e^{mx} - \sqrt{b}}{\sqrt{a}\,e^{mx} + \sqrt{b}} \\ \frac{1}{m\sqrt{ab}} & \tanh^{-1}\left(\frac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \\ \frac{-1}{m\sqrt{ab}} & \coth^{-1}\left(\frac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \end{array} \right.$$

These examples also show line-drawing, built-up characters like braces and radicals, and a spectrum of positioning problems. (Section 6 shows what a user has to type to produce these on our system.)

2. Photocomposition

Photocomposition techniques can be used to solve some of the problems of typesetting mathematics. A phototypesetter is a device which exposes a piece of photographic paper or film, placing characters wherever they are wanted. The Graphic Systems phototypesetter[2] on the UNIX operating system[3] works by shining light through a character stencil. The character is made the right size by lenses, and the light beam directed by fiber optics to the desired place on a piece of photographic paper. The exposed paper is developed and typically used in some form of photo-offset reproduction.

On UNIX, the phototypesetter is driven by a formatting program called TROFF [4]. TROFF was designed for setting running text. It also provides all of the facilities that one needs for doing mathematics, such as arbitrary horizontal and vertical motions, line-drawing, size changing, but the syntax for describing these special operations is difficult to learn, and difficult even for experienced users to type correctly.

For this reason we decided to use TROFF as an "assembly language," by designing a language for describing mathematical expressions, and compiling it into TROFF.

3. Language Design

The fundamental principle upon which we based our language design is that the language should be easy to use by people (for example, secretaries) who know neither mathematics nor typesetting.

This principle implies several things. First, "normal" mathematical conventions about operator precedence, parentheses, and the like cannot be used, for to give special meaning to such characters means that the user has to understand what he or she is typing. Thus the language should not assume, for instance, that parentheses are always balanced, for they are not in the half-open interval $(a,b]$. Nor should it assume that that $\sqrt{a+b}$ can be replaced by $(a+b)^{1/2}$, or that $1/(1-x)$ is better written as $\frac{1}{1-x}$ (or vice versa).

Second, there should be relatively few rules, keywords, special symbols and operators, and the like. This keeps the language easy to learn and remember. Furthermore, there should be few exceptions to the rules that do exist: if something works in one situation, it should work everywhere. If a variable can have a subscript, then a subscript can have a subscript, and so on without limit.

Third, "standard" things should happen automatically. Someone who types "x=y+z+1" should get "$x=y+z+1$". Subscripts and superscripts should automatically be printed in an appropriately smaller size, with no special intervention. Fraction bars have to be made the right length and positioned at the right height. And so on. Indeed a mechanism for overriding default actions has to exist, but its application is the exception, not the rule.

We assume that the typist has a reasonable picture (a two-dimensional representation) of the desired final form, as might be handwritten by the author of a paper. We also assume that the input is typed on a computer terminal much like an ordinary typewriter. This implies an input alphabet of perhaps 100 characters, none of them special.

A secondary, but still important, goal in our design was that the system should be easy to implement, since neither of the authors had any desire to make a long-term project of it. Since our design was not firm, it was also necessary that the program be easy to change at any time.

To make the program easy to build and to change, and to guarantee regularity ("it should work everywhere"), the language is defined by a context-free grammar, described in Section 5. The compiler for the language was built using a compiler-compiler.

A priori, the grammar/compiler-compiler approach seemed the right thing to do. Our subsequent experience leads us to believe that any other course would have been folly. The original language was designed in a few days. Construction of a working system sufficient to try significant examples required perhaps a person-month. Since then, we have spent a modest amount of additional time over several years tuning, adding facilities, and occasionally changing the language as users make criticisms and suggestions.

We also decided quite early that we would let TROFF do our work for us whenever possible. TROFF is quite a powerful program, with a macro facility, text and arithmetic variables, numerical computation and testing, and conditional branching. Thus we have been able to avoid writing a lot of mundane but tricky software. For example, we store no text strings, but simply pass them on to TROFF. Thus we avoid having to write a storage management package. Furthermore, we have been able to isolate ourselves from most details of the particular device and character set currently in use. For example, we let TROFF compute the widths of all strings of characters; we need know nothing about them.

A third design goal is special to our environment. Since our program is only useful for typesetting mathematics, it is necessary that it interface cleanly with the underlying typesetting language for the benefit of users who want to set intermingled mathematics and text (the usual case). The standard mode of operation is that when a document is typed, mathematical expressions are input as part of the text, but marked by user settable delimiters. The program reads this input and treats as comments those things which are not mathematics, simply passing them through untouched. At the same time it converts the mathematical input into the necessary TROFF commands. The resulting ioutput is passed directly to TROFF where the comments and the mathematical parts both become text and/or TROFF commands.

4. The Language

We will not try to describe the language precisely here; interested readers may refer to the appendix for more details. Throughout this section, we will write expressions exactly as they are handed to the typesetting program (hereinafter called "EQN"), except that we won't show the delimiters that the user types to mark the beginning and end of the expression. The interface between EQN and TROFF is described at the end of this section.

As we said, typing x=y+z+1 should produce $x=y+z+1$, and indeed it does. Variables are made italic, operators and digits become roman, and normal spacings between letters and operators are altered slightly to give a more pleasing appearance.

Input is free-form. Spaces and new lines in the input are used by EQN to separate pieces of the input; they are not used to create space in the output. Thus

```
x  =  y
   + z + 1
```

also gives $x=y+z+1$. Free-form input is easier to type initially; subsequent editing is also easier, for an expression may be typed as many short lines.

Extra white space can be forced into the output by several characters of various sizes. A tilde "~" gives a space equal to the normal word spacing in text; a circumflex gives half this much, and a tab charcter spaces to the next tab stop.

Spaces (or tildes, etc.) also serve to delimit pieces of the input. For example, to get

$$f(t)=2\pi\int\sin(\omega t)dt$$

we write

```
f(t) = 2 pi int sin ( omega t )dt
```

Here spaces are *necessary* in the input to indicate that *sin, pi, int,* and *omega* are special, and potentially worth special treatment. EQN looks up each such string of characters in a table, and if appropriate gives it a translation. In this case, *pi* and *omega* become their greek equivalents, *int* becomes the integral sign (which must be moved down and enlarged so it looks "right"), and *sin* is made roman, following conventional mathematical practice. Parentheses, digits and operators are automatically made roman wherever found.

Fractions are specified with the keyword *over:*

```
a+b over c+d+e = 1
```

produces

$$\frac{a+b}{c+d+e}=1$$

Similarly, subscripts and superscripts are introduced by the keywords *sub* and *sup:*

$$x^2+y^2=z^2$$

is produced by

```
x sup 2 + y sup 2 = z sup 2
```

The spaces after the 2's are necessary to mark the end of the superscripts; similarly the keyword *sup* has to be marked off by spaces or some equivalent delimiter. The return to the proper baseline is automatic. Multiple levels of subscripts or superscripts are of course allowed: "x sup y sup z" is x^{y^z}. The construct "something *sub* something *sup* something" is recognized as a special case, so "x sub i sup 2" is x_i^2 instead of x_{i^2}.

More complicated expressions can now be formed with these primitives:

$$\frac{\partial^2 f}{\partial x^2}=\frac{x^2}{a^2}+\frac{y^2}{b^2}$$

is produced by

```
{partial sup 2 f} over {partial x sup 2} =
x sup 2 over a sup 2 + y sup 2 over b sup 2
```

Braces {} are used to group objects together; in this case they indicate unambiguously what goes over what on the left-hand side of the expression. The language defines the precedence of *sup* to be higher than that of *over,* so no braces are needed to get the correct association on the right side. Braces can always be used when in doubt about precedence.

The braces convention is an example of

the power of using a recursive grammar to define the language. It is part of the language that if a construct can appear in some context, then *any expression* in braces can also occur in that context.

There is a *sqrt* operator for making square roots of the appropriate size: "sqrt a+b" produces $\sqrt{a+b}$, and

```
x = {-b +- sqrt{b sup 2 -4ac}} over 2a
```

is

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Since large radicals look poor on our typesetter, *sqrt* is not useful for tall expressions.

Limits on summations, integrals and similar constructions are specified with the keywords *from* and *to.* To get

$$\sum_{i=0}^{\infty} x_i \to 0$$

we need only type

```
sum from i=0 to inf x sub i -> 0
```

Centering and making the Σ big enough and the limits smaller are all automatic. The *from* and *to* parts are both optional, and the central part (e.g., the Σ) can in fact be anything:

```
lim from {x -> pi /2} ( tan~x) = inf
```

is

$$\lim_{x\to\pi/2}(\tan x)=\infty$$

Again, the braces indicate just what goes into the *from* part.

There is a facility for making braces, brackets, parentheses, and vertical bars of the right height, using the keywords *left* and *right:*

```
left [ x+y over 2a right ]~=~1
```

makes

$$\left[\frac{x+y}{2a}\right] = 1$$

A *left* need not have a corresponding *right,* as we shall see in the next example. Any characters may follow *left* and *right,* but generally only various parentheses and bars are meaningful.

Big brackets, etc., are often used with another facility, called *piles,* which make vertical piles of objects. For example, to get

$$sign\,(x) \equiv \left\{ \begin{array}{rl} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{array}\right.$$

we can type

```
sign (x) ~==~ left {
  rpile {1 above 0 above -1}
  ~~lpile {if above if above if}
  ~~lpile {x>0 above x=0 above x<0}
```

The construction "left {" makes a left brace big enough to enclose the "rpile {...}", which is a right-justified pile of "above ... above ...". "lpile" makes a left-justified pile. There are also centered piles. Because of the recursive language definition, a pile can contain any number of elements; any element of a pile can of course contain piles.

Although EQN makes a valiant attempt to use the right sizes and fonts, there are times when the default assumptions are simply not what is wanted. For instance the italic *sign* in the previous example would conventionally be in roman. Slides and transparencies often require larger characters than normal text. Thus we also provide size and font changing commands: "size 12 bold {A~x~=~y}" will produce $\mathbf{A\ x = y}$. *Size* is followed by a number representing a character size in points. (One point is 1/72 inch; this paper is set in 9 point type.)

If necessary, an input string can be quoted in "...", which turns off grammatical significance, and any font or spacing changes that might otherwise be done on it. Thus we can say

```
lim~ roman "sup" ~x sub n = 0
```

to ensure that the supremum doesn't become a superscript:

$$\lim \mathrm{sup}\ x_n=0$$

Diacritical marks, long a problem in traditional typesetting, are straightforward:

$$\underset{\cdot}{x}+\hat{x}+\tilde{y}+\hat{X}+\ddot{Y}=\overline{z+Z}$$

is made by typing

```
x dot under + x hat + y tilde
+ X hat + Y dotdot = z+Z bar
```

There are also facilities for globally changing default sizes and fonts, for example for making viewgraphs or for setting chemical equations. The language allows for matrices, and for lining up equations at the same horizontal position.

Finally, there is a definition facility, so a user can say

```
define name "..."
```

at any time in the document; henceforth, any occurrence of the token "name" in an expression will be expanded into whatever was inside the double quotes in its definition. This lets users tailor the language to their own specifications, for it is quite possible to redefine keywords like *sup* or *over.* Section 6 shows an example of definitions.

The EQN preprocessor reads intermixed text and equations, and passes its output to TROFF. Since TROFF uses lines beginning with a period as control words (e.g., ".ce" means "center the next output line"), EQN uses the sequence ".EQ" to mark the beginning of an equation and ".EN" to mark the end. The ".EQ" and ".EN" are passed through to TROFF untouched, so they can also be used by a knowledgeable user to center equations, number them automatically, etc. By default, however, ".EQ" and ".EN" are simply ignored by TROFF, so by default equations are printed in-line.

".EQ" and ".EN" can be supplemented by TROFF commands as desired; for example, a centered display equation can be produced with the input:

```
.ce
.EQ
x sub i = y sub i ...
.EN
```

Since it is tedious to type ".EQ" and ".EN" around very short expressions (single letters, for instance), the user can also define two characters to serve as the left and right delimiters of expressions. These characters are recognized anywhere in subsequent text. For example if the left and right delimiters have both been set to "#", the input:

Let #x sub i#, #y# and #alpha# be positive

produces:

Let x_i, y and α be positive

Running a preprocessor is strikingly easy on UNIX. To typeset text stored in file "f", one issues the command:

```
eqn f | troff
```

The vertical bar connects the output of one process (EQN) to the input of another (TROFF).

5. Language Theory

The basic structure of the language is not a particularly original one. Equations are pictured as a set of "boxes," pieced together in various ways. For example, something with a subscript is just a box followed by another box moved downward and shrunk by an appropriate amount. A fraction is just a box centered above another box, at the right altitude, with a line of correct length drawn between them.

The grammar for the language is shown below. For purposes of exposition, we have collapsed some productions. In the original grammar, there are about 70 productions, but many of these are simple ones used only to guarantee that some keyword is recognized early enough in the parsing process. Symbols in capital letters are terminal symbols; lower case symbols are non-terminals, i.e., syntactic categories. The vertical bar | indicates an alternative; the brackets [ ] indicate optional material. A TEXT is a string of non-blank characters or any string inside double quotes; the other terminal symbols represent literal occurrences of the corresponding keyword.

```
eqn  : box | eqn box
box  : text
     | { eqn }
     | box OVER box
     | SQRT box
     | box SUB box | box SUP box
     | [ L | C | R ]PILE { list }
     | LEFT text eqn [ RIGHT text ]
     | box [ FROM box ] [ TO box ]
     | SIZE text box
     | [ROMAN | BOLD | ITALIC] box
     | box [HAT | BAR | DOT | DOTDOT | TILDE]
     | DEFINE text text
list : eqn | list ABOVE eqn
text : TEXT
```

The grammar makes it obvious why there are few exceptions. For example, the observation that something can be replaced by a more complicated something in braces is implicit in the productions:

```
eqn : box | eqn box
box : text | { eqn }
```

Anywhere a single character could be used, *any* legal construction can be used.

Clearly, our grammar is highly ambiguous. What, for instance, do we do with the input

a over b over c ?

Is it

{a over b} over c

or is it

a over {b over c} ?

To answer questions like this, the grammar is supplemented with a small set of rules that describe the precedence and associativity of operators. In particular, we specify (more or less arbitrarily) that *over* associates to the left, so the first alternative above is the one chosen. On the other hand, *sub* and *sup* bind to the right, because this is closer to standard mathematical practice. That is, we assume x^{a^b} is $x^{(a^b)}$, not $(x^a)^b$.

The precedence rules resolve the ambiguity in a construction like

a sup 2 over b

We define *sup* to have a higher precedence than *over*, so this construction is parsed as $\frac{a^2}{b}$ instead of $a^{\frac{2}{b}}$.

Naturally, a user can always force a particular parsing by placing braces around expressions.

The ambiguous grammar approach seems to be quite useful. The grammar we use is small enough to be easily understood, for it contains none of the productions that would be normally used for resolving ambiguity. Instead the supplemental information about precedence and associativity (also small enough to be understood) provides the compiler-compiler with the information it needs to make a fast, deterministic parser for the specific language we want. When the language is supplemented by the disambiguating rules, it is in fact LR(1) and thus easy to parse[5].

The output code is generated as the input is scanned. Any time a production of the grammar is recognized, (potentially) some TROFF commands are output. For example, when the lexical analyzer reports that it has found a TEXT (i.e., a string of contiguous characters), we have recognized the production:

```
text    : TEXT
```

The translation of this is simple. We generate a local name for the string, then hand the name and the string to TROFF, and let TROFF perform the storage management. All we save is the name of the string, its height, and its baseline.

As another example, the translation associated with the production

```
box    : box OVER box
```

is:

```
Width of output box =
  slightly more than largest input width
Height of output box =
  slightly more than sum of input heights
Base of output box =
  slightly more than height of bottom input box
String describing output box =
  move down;
  move right enough to center bottom box;
  draw bottom box (i.e., copy string for bottom box);
  move up; move left enough to center top box;
  draw top box (i.e., copy string for top box);
  move down and left; draw line full width;
  return to proper base line.
```

Most of the other productions have equally simple semantic actions. Picturing the output as a set of properly placed boxes makes the right sequence of positioning commands quite obvious. The main difficulty is in finding the right numbers to use for esthetically pleasing positioning.

With a grammar, it is usually clear how to extend the language. For instance, one of our users suggested a TENSOR operator, to make constructions like

$${}^{l}_{m}\mathbf{T}^{k\,j}_{n\,i}$$

Grammatically, this is easy: it is sufficient to add a production like

```
box    : TENSOR { list }
```

Semantically, we need only juggle the boxes to the right places.

6. Experience

There are really three aspects of interest—how well EQN sets mathematics, how well it satisfies its goal of being "easy to use," and how easy it was to build.

The first question is easily addressed. This entire paper has been set by the program. Readers can judge for themselves whether it is good enough for their purposes. One of our users commented that although the output is not as good as the best hand-set material, it is still better than average, and much better than the worst. In any case, who cares? Printed books cannot compete with the birds and flowers of illuminated manuscripts on esthetic grounds, either, but they have some clear economic advantages.

Some of the deficiencies in the output could be cleaned up with more work on our part. For example, we sometimes leave too much space between a roman letter and an italic one. If we were willing to keep track of the fonts involved, we could do this better more of the time.

Some other weaknesses are inherent in our output device. It is hard, for instance, to draw a line of an arbitrary length without getting a perceptible overstrike at one end.

As to ease of use, at the time of writing, the system has been used by two distinct groups. One user population consists of mathematicians, chemists, physicists, and computer scientists. Their typical reaction has been something like:

(1) It's easy to write, although I make the following mistakes...

(2) How do I do...?

(3) It botches the following things.... Why don't you fix them?

(4) You really need the following features...

The learning time is short. A few minutes gives the general flavor, and typing a page or two of a paper generally uncovers most of the misconceptions about how it works.

The second user group is much larger, the secretaries and mathematical typists who were the original target of the system. They tend to be enthusiastic converts. They find the language easy to learn (most are largely self-taught), and have little trouble producing the output they want. They are of course less critical of the esthetics of their output than users trained in mathematics. After a transition period, most find using a computer more interesting than a regular typewriter.

The main difficulty that users have seems to be remembering that a blank is a delimiter; even experienced users use blanks where they shouldn't and omit them when they are needed. A common instance is typing

```
f(x sub i)
```

which produces

$$f(x_{i)}$$

instead of

$$f(x_i)$$

Since the EQN language knows no mathematics, it cannot deduce that the right parenthesis is not part of the subscript.

The language is somewhat prolix, but this doesn't seem excessive considering how much is being done, and it is certainly more compact than the corresponding TROFF commands. For example, here is the source for the continued fraction expression in Section 1 of this paper:

```
a sub 0 + b sub 1 over
  {a sub 1 + b sub 2 over
    {a sub 2 + b sub 3 over
      {a sub 3 + ... }}}
```

This is the input for the large integral of Section 1; notice the use of definitions:

```
define emx "{e sup mx}"
define mab "{m sqrt ab}"
define sa "{sqrt a}"
define sb "{sqrt b}"
int dx over {a emx - be sup -mx} ~=~
left { lpile {
    1 over {2 mab} ~log~
        {sa emx - sb} over {sa emx + sb}
  above
    1 over mab ~ tanh sup -1 ( sa over sb emx )
  above
    -1 over mab ~ coth sup -1 ( sa over sb emx )
}
```

As to ease of construction, we have already mentioned that there are really only a few person-months invested. Much of this time has gone into two things—fine-tuning (what is the most esthetically pleasing space to use between the numerator and denominator of a fraction?), and changing things found deficient by our users (shouldn't a tilde be a delimiter?).

The program consists of a number of small, essentially unconnected modules for code generation, a simple lexical analyzer, a canned parser which we did not have to write, and some miscellany associated with input files and the macro facility. The program is now about 1600 lines of C [6], a high-level language reminiscent of BCPL. About 20 percent of these lines are "print" statements, generating the output code.

The semantic routines that generate the actual TROFF commands can be changed to accommodate other formatting languages and devices. For example, in less than 24 hours, one of us changed the entire semantic package to drive NROFF, a variant of TROFF, for typesetting mathematics on teletypewriter devices capable of reverse line motions. Since many potential users do not have access to a typesetter, but still have to type mathematics, this provides a way to get a typed version of the final output which is close enough for debugging purposes, and sometimes even for ultimate use.

7. Conclusions

We think we have shown that it is possible to do acceptably good typesetting of mathematics on a phototypesetter, with an input language that is easy to learn and use and that satisfies many users' demands. Such a package can be implemented in short order, given a compiler-compiler and a decent typesetting program underneath.

Defining a language, and building a compiler for it with a compiler-compiler seems like the only sensible way to do business. Our experience with the use of a grammar and a compiler-compiler has been uniformly favorable. If we had written everything into code directly, we would have been locked into our original design. Furthermore, we would have never been sure where the exceptions and special cases were. But because we have a grammar, we can change our minds readily and still be reasonably sure that if a construction works in one place it will work everywhere.

Acknowledgements

We are deeply indebted to J. F. Ossanna, the author of TROFF, for his willingness to modify TROFF to make our task easier and for his continuous assistance during the development of our program. We are also grateful to A. V. Aho for help with language theory, to S. C. Johnson for aid with the compiler-compiler, and to our early users A. V. Aho, S. I. Feldman, S. C. Johnson, R. W. Hamming, and M. D. McIlroy for their constructive criticisms.

References

[1] *A Manual of Style*, 12th Edition. University of Chicago Press, 1969. p 295.

[2] *Model C/A/T Phototypesetter*. Graphic Systems, Inc., Hudson, N. H.

[3] Ritchie, D. M., and Thompson, K. L., "The UNIX time-sharing system." *Comm. ACM 17*, 7 (July 1974), 365-375.

[4] Ossanna, J. F., TROFF User's Manual. Bell Laboratories Computing Science Technical Report 54, 1977.

[5] Aho, A. V., and Johnson, S. C., "LR Parsing." *Comp. Surv. 6*, 2 (June 1974), 99-124.

[6] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Prentice-Hall, Inc., 1978.

Typesetting Mathematics — User's Guide (Second Edition)

Brian W. Kernighan and Lorinda L. Cherry

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This is the user's guide for a system for typesetting mathematics, using the phototypesetters on the UNIX† and GCOS operating systems.

Mathematical expressions are described in a language designed to be easy to use by people who know neither mathematics nor typesetting. Enough of the language to set in-line expressions like $\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$ or display equations like

$$G(z) = e^{\ln G(z)} = \exp\left(\sum_{k \geq 1} \frac{S_k z^k}{k} \right) = \prod_{k \geq 1} e^{S_k z^k / k}$$

$$= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots \right) \left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \cdots \right) \cdots$$

$$= \sum_{m \geq 0} \left(\sum_{\substack{k_1, k_2, \ldots, k_m \geq 0 \\ k_1 + 2k_2 + \cdots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \, \frac{S_2^{k_2}}{2^{k_2} k_2!} \cdots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right) z^m$$

can be learned in an hour or so.

The language interfaces directly with the phototypesetting language TROFF, so mathematical expressions can be embedded in the running text of a manuscript, and the entire document produced in one process. This user's guide is an example of its output.

The same language may be used with the UNIX formatter NROFF to set mathematical expressions on DASI and GSI terminals and Model 37 teletypes.

August 15, 1978

†UNIX is a Trademark of Bell Laboratories.

Typesetting Mathematics — User's Guide (Second Edition)

Brian W. Kernighan and Lorinda L. Cherry

Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

EQN is a program for typesetting mathematics on the Graphics Systems phototypesetters on UNIX and GCOS. The EQN language was designed to be easy to use by people who know neither mathematics nor typesetting. Thus EQN knows relatively little about mathematics. In particular, mathematical symbols like +, −, ×, parentheses, and so on have no special meanings. EQN is quite happy to set garbage (but it will look good).

EQN works as a preprocessor for the typesetter formatter, TROFF[1], so the normal mode of operation is to prepare a document with both mathematics and ordinary text interspersed, and let EQN set the mathematics while TROFF does the body of the text.

On UNIX, EQN will also produce mathematics on DASI and GSI terminals and on Model 37 teletypes. The input is identical, but you have to use the programs NEQN and NROFF instead of EQN and TROFF. Of course, some things won't look as good because terminals don't provide the variety of characters, sizes and fonts that a typesetter does, but the output is usually adequate for proofreading.

To use EQN on UNIX,

```
eqn files | troff
```

GCOS use is discussed in section 26.

2. Displayed Equations

To tell EQN where a mathematical expression begins and ends, we mark it with lines beginning .EQ and .EN. Thus if you type the lines

```
.EQ
x=y+z
.EN
```

your output will look like

$$x=y+z$$

The .EQ and .EN are copied through untouched; they are not otherwise processed by EQN. This means that you have to take care of things like centering, numbering, and so on yourself. The most common way is to use the TROFF and NROFF macro package package '−ms' developed by M. E. Lesk[3], which allows you to center, indent, left-justify and number equations.

With the '−ms' package, equations are centered by default. To left-justify an equation, use .EQ L instead of .EQ. To indent it, use .EQ I. Any of these can be followed by an arbitrary 'equation number' which will be placed at the right margin. For example, the input

```
.EQ I (3.1a)
x = f(y/2) + y/2
.EN
```

produces the output

$$x=f(y/2)+y/2 \qquad (3.1a)$$

There is also a shorthand notation so in-line expressions like π_i^2 can be entered without .EQ and .EN. We will talk about it in section 19.

3. Input spaces

Spaces and newlines within an expression are thrown away by EQN. (Normal text is left absolutely alone.) Thus between .EQ and .EN,

```
x=y+z
```

and

```
    x = y + z
```

and

```
    x  =  y
       + z
```

and so on all produce the same output

$$x{=}y{+}z$$

You should use spaces and newlines freely to make your input equations readable and easy to edit. In particular, very long lines are a bad idea, since they are often hard to fix if you make a mistake.

4. Output spaces

To force extra spaces into the *output,* use a tilde "~" for each space you want:

```
    x~=~y~+~z
```

gives

$$x ~=~ y ~+~ z$$

You can also use a circumflex "^", which gives a space half the width of a tilde. It is mainly useful for fine-tuning. Tabs may also be used to position pieces of an expression, but the tab stops must be set by TROFF commands.

5. Symbols, Special Names, Greek

EQN knows some mathematical symbols, some mathematical names, and the Greek alphabet. For example,

```
    x=2 pi int sin ( omega t)dt
```

produces

$$x{=}2\pi\int\sin(\omega t)\,dt$$

Here the spaces in the input are **necessary** to tell EQN that *int, pi, sin* and *omega* are separate entities that should get special treatment. The *sin,* digit 2, and parentheses are set in roman type instead of italic; *pi* and *omega* are made Greek; and *int* becomes the integral sign.

When in doubt, leave spaces around separate parts of the input. A *very* common error is to type *f(pi)* without leaving spaces on both sides of the *pi.* As a result, EQN does not recognize *pi* as a special word, and it appears as $f(pi)$ instead of $f(\pi)$.

A complete list of EQN names appears in section 23. Knowledgeable users can also use TROFF four-character names for anything EQN doesn't know about, like \(*bs* for the Bell System sign Ⓑ.

6. Spaces, Again

The only way EQN can deduce that some sequence of letters might be special is if that sequence is separated from the letters on either side of it. This can be done by surrounding a special word by ordinary spaces (or tabs or newlines), as we did in the previous section.

You can also make special words stand out by surrounding them with tildes or circumflexes:

```
    x~=~2~pi~int~sin~(~omega~t~)~dt
```

is much the same as the last example, except that the tildes not only separate the magic words like *sin, omega,* and so on, but also add extra spaces, one space per tilde:

$$x ~=~ 2 ~\pi~ \int ~\sin~ (~\omega~ t ~) ~dt$$

Special words can also be separated by braces { } and double quotes "...", which have special meanings that we will see soon.

7. Subscripts and Superscripts

Subscripts and superscripts are obtained with the words *sub* and *sup.*

```
    x sup 2 + y sub k
```

gives

$$x^2{+}y_k$$

EQN takes care of all the size changes and vertical motions needed to make the output look right. The words *sub* and *sup* must be surrounded by spaces; *x sub2* will give you *xsub*2 instead of x_2. Furthermore, don't forget to leave a space (or a tilde, etc.) to mark the end of a subscript or superscript. A common error is to say something like

```
    y = (x sup 2)+1
```

which causes

$$y{=}(x^{2)+1}$$

instead of the intended

$$y{=}(x^2){+}1$$

Subscripted subscripts and superscripted superscripts also work:

x sub i sub 1

is

$$x_{i_1}$$

A subscript and superscript on the same thing are printed one above the other if the subscript comes *first:*

x sub i sup 2

is

$$x_i^2$$

Other than this special case, *sub* and *sup* group to the right, so *x sup y sub z* means x^{y_z}, not $x^y{}_z$.

8. Braces for Grouping

Normally, the end of a subscript or superscript is marked simply by a blank (or tab or tilde, etc.) What if the subscript or superscript is something that has to be typed with blanks in it? In that case, you can use the braces { and } to mark the beginning and end of the subscript or superscript:

e sup {i omega t}

is

$$e^{i\omega t}$$

Rule: Braces can *always* be used to force EQN to treat something as a unit, or just to make your intent perfectly clear. Thus:

x sub {i sub 1} sup 2

is

$$x_{i_1}^2$$

with braces, but

x sub i sub 1 sup 2

is

$$x_{i_1^2}$$

which is rather different.

Braces can occur within braces if necessary:

e sup {i pi sup {rho +1}}

is

$$e^{i\pi^{\rho+1}}$$

The general rule is that anywhere you could use some single thing like *x*, you can use an arbitrarily complicated thing if you enclose it in braces. EQN will look after all the details of positioning it and making it the right size.

In all cases, make sure you have the right number of braces. Leaving one out or adding an extra will cause EQN to complain bitterly.

Occasionally you will have to print braces. To do this, enclose them in double quotes, like "{". Quoting is discussed in more detail in section 14.

9. Fractions

To make a fraction, use the word *over:*

a+b over 2c =1

gives

$$\frac{a+b}{2c}=1$$

The line is made the right length and positioned automatically. Braces can be used to make clear what goes over what:

{alpha + beta} over {sin (x)}

is

$$\frac{\alpha+\beta}{\sin(x)}$$

What happens when there is both an *over* and a *sup* in the same expression? In such an apparently ambiguous case, EQN does the *sup* before the *over*, so

−b sup 2 over pi

is $\frac{-b^2}{\pi}$ instead of $-b^{\frac{2}{\pi}}$ The rules which decide which operation is done first in cases like this are summarized in section 23. When in doubt, however, *use braces* to make clear what goes with what.

10. Square Roots

To draw a square root, use *sqrt:*

sqrt a+b + 1 over sqrt {ax sup 2 +bx+c}

is

$$\sqrt{a+b}+\frac{1}{\sqrt{ax^2+bx+c}}$$

Warning — square roots of tall quantities look lousy, because a root-sign big enough to cover the quantity is too dark and heavy:

sqrt {a sup 2 over b sub 2}

is

$$\sqrt{\frac{a^2}{b_2}}$$

Big square roots are generally better written as something to the power ½:

$$(a^2/b_2)^{1/2}$$

which is

(a sup 2 /b sub 2) sup half

11. Summation, Integral, Etc.

Summations, integrals, and similar constructions are easy:

sum from i=0 to {i= inf} x sup i

produces

$$\sum_{i=0}^{i=\infty} x^i$$

Notice that we used braces to indicate where the upper part $i=\infty$ begins and ends. No braces were necessary for the lower part $i=0$, because it contained no blanks. The braces will never hurt, and if the *from* and *to* parts contain any blanks, you must use braces around them.

The *from* and *to* parts are both optional, but if both are used, they have to occur in that order.

Other useful characters can replace the *sum* in our example:

int prod union inter

become, respectively,

$$\int \quad \prod \quad \bigcup \quad \bigcap$$

Since the thing before the *from* can be anything, even something in braces, *from-to* can often be used in unexpected ways:

lim from {n −> inf} x sub n =0

is

$$\lim_{n \to \infty} x_n = 0$$

12. Size and Font Changes

By default, equations are set in 10-point type (the same size as this guide), with standard mathematical conventions to determine what characters are in roman and what in italic. Although EQN makes a valiant attempt to use esthetically pleasing sizes and fonts, it is not perfect. To change sizes and fonts, use *size n* and *roman, italic, bold* and *fat*. Like *sub* and *sup*, size and font changes affect only the thing that follows them, and revert to the normal situation at the end of it. Thus

bold x y

is

$$\mathbf{x}y$$

and

size 14 bold x = y +
  size 14 {alpha + beta}

gives

$$\mathbf{x}=y+\alpha+\beta$$

As always, you can use braces if you want to affect something more complicated than a single letter. For example, you can change the size of an entire equation by

size 12 { ... }

Legal sizes which may follow *size* are 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, 36. You can also change the size *by* a given amount; for example, you can say *size +2* to make the size two points bigger, or *size −3* to make it three points smaller. This has the advantage that you don't have to know what the current size is.

If you are using fonts other than roman, italic and bold, you can say *font X* where *X* is a one character TROFF name or number for the font. Since EQN is tuned for roman, italic and bold, other fonts may not give quite as good an appearance.

The *fat* operation takes the current font and widens it by overstriking: *fat grad* is ∇ and *fat {x sub i}* is $\boldsymbol{x_i}$.

If an entire document is to be in a non-standard size or font, it is a severe nuisance to have to write out a size and font change for each equation. Accordingly, you can set a "global" size or font which thereafter affects all equations. At the beginning of any equation, you might say, for instance,

```
.EQ
gsize 16
gfont R
...
.EN
```

to set the size to 16 and the font to roman thereafter. In place of R, you can use any of the TROFF font names. The size after *gsize* can be a relative change with + or −.

Generally, *gsize* and *gfont* will appear at the beginning of a document but they can also appear thoughout a document: the global font and size can be changed as often as needed. For example, in a footnote‡ you will typically want the size of equations to match the size of the footnote text, which is two points smaller than the main text. Don't forget to reset the global size at the end of the footnote.

13. Diacritical Marks

To get funny marks on top of letters, there are several words:

x dot	$\dot{x}$
x dotdot	$\ddot{x}$
x hat	$\hat{x}$
x tilde	$\tilde{x}$
x vec	$\vec{x}$
x dyad	$\overleftrightarrow{x}$
x bar	$\bar{x}$
x under	$\underline{x}$

The diacritical mark is placed at the right height. The *bar* and *under* are made the right length for the entire construct, as in $\overline{x+y+z}$; other marks are centered.

14. Quoted Text

Any input entirely within quotes ("...") is not subject to any of the font changes and spacing adjustments normally done by the equation setter. This provides a way to do your own spacing and adjusting if needed:

```
italic "sin(x)" + sin (x)
```

is

sin(x)+sin(x)

Quotes are also used to get braces and other EQN keywords printed:

```
"{ size alpha }"
```

is

{ *size alpha* }

and

```
roman "{ size alpha }"
```

is

{ size alpha }

The construction "" is often used as a place-holder when grammatically EQN needs something, but you don't actually want anything in your output. For example, to make ^{2}He, you can't just type *sup 2 roman He* because a *sup* has to be a superscript *on* something. Thus you must say

```
"" sup 2 roman He
```

To get a literal quote use "\"". TROFF characters like \(*bs* can appear unquoted, but more complicated things like horizontal and vertical motions with *h* and *v* should always be quoted. (If you've never heard of *h* and *v*, ignore this section.)

15. Lining Up Equations

Sometimes it's necessary to line up a series of equations at some horizontal position, often at an equals sign. This is done with two operations called *mark* and *lineup*.

The word *mark* may appear once at any place in an equation. It remembers the horizontal position where it appeared. Successive equations can contain one occurrence of the word *lineup*. The place where *lineup* appears is made to line up with the place marked by the previous *mark* if at all possible. Thus, for example, you can say

‡Like this one, in which we have a few random expressions like x_i and π^2. The sizes for these were set by the command *gsize −2*.

```
.EQ I
x+y mark = z
.EN
.EQ I
x lineup = 1
.EN
```

to produce

$$x+y=z$$

$$x=1$$

For reasons too complicated to talk about, when you use EQN and '–ms', use either .EQ I or .EQ L. mark and *lineup* don't work with centered equations. Also bear in mind that *mark* doesn't look ahead;

```
x mark =1
...
x+y lineup =z
```

isn't going to work, because there isn't room for the $x+y$ part after the *mark* remembers where the x is.

16. Big Brackets, Etc.

To get big brackets [ ], braces { }, parentheses ( ), and bars || around things, use the *left* and *right* commands:

```
left { a over b + 1 right }
~=~ left ( c over d right )
+ left [ e right ]
```

is

$$\left\{ \frac{a}{b}+1 \right\} = \left(\frac{c}{d} \right) + \left[e \right]$$

The resulting brackets are made big enough to cover whatever they enclose. Other characters can be used besides these, but the are not likely to look very good. One exception is the *floor* and *ceiling* characters:

```
left floor x over y right floor
<= left ceiling a over b right ceiling
```

produces

$$\left\lfloor \frac{x}{y} \right\rfloor \leqslant \left\lceil \frac{a}{b} \right\rceil$$

Several warnings about brackets are in order. First, braces are typically bigger than brackets and parentheses, because they are made up of three, five, seven, etc., pieces, while brackets can be made up of two, three, etc. Second, big left and right parentheses often look poor, because the character set is poorly designed.

The *right* part may be omitted: a "left something" need not have a corresponding "right something". If the *right* part is omitted, put braces around the thing you want the left bracket to encompass. Otherwise, the resulting brackets may be too large.

If you want to omit the *left* part, things are more complicated, because technically you can't have a *right* without a corresponding *left*. Instead you have to say

```
left "" ..... right )
```

for example. The *left* "" means a "left nothing". This satisfies the rules without hurting your output.

17. Piles

There is a general facility for making vertical piles of things; it comes in several flavors. For example:

```
A ~=~ left [
  pile { a above b above c }
  ~~ pile { x above y above z }
right ]
```

will make

$$A = \left[\begin{matrix} a & x \\ b & y \\ c & z \end{matrix} \right]$$

The elements of the pile (there can be as many as you want) are centered one above another, at the right height for most purposes. The keyword *above* is used to separate the pieces; braces are used around the entire list. The elements of a pile can be as complicated as needed, even containing more piles.

Three other forms of pile exist: *lpile* makes a pile with the elements left-justified; *rpile* makes a right-justified pile; and *cpile* makes a centered pile, just like *pile*. The vertical spacing between the pieces is somewhat larger for *l-*, *r-* and *cpiles* than it is for ordinary piles.

```
roman sign (x)~=~
left {
   lpile {1 above 0 above -1}
   ~~ lpile
   {if~x>0 above if~x=0 above if~x<0}
```

makes

$$\mathrm{sign}(x) = \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{cases}$$

Notice the left brace without a matching right one.

18. Matrices

It is also possible to make matrices. For example, to make a neat array like

$$\begin{matrix} x_i & x^2 \\ y_i & y^2 \end{matrix}$$

you have to type

```
matrix {
  ccol { x sub i above y sub i }
  ccol { x sup 2 above y sup 2 }
}
```

This produces a matrix with two centered columns. The elements of the columns are then listed just as for a pile, each element separated by the word *above*. You can also use *lcol* or *rcol* to left or right adjust columns. Each column can be separately adjusted, and there can be as many columns as you like.

The reason for using a matrix instead of two adjacent piles, by the way, is that if the elements of the piles don't all have the same height, they won't line up properly. A matrix forces them to line up, because it looks at the entire structure before deciding what spacing to use.

A word of warning about matrices — *each column must have the same number of elements in it.* The world will end if you get this wrong.

19. Shorthand for In-line Equations

In a mathematical document, it is necessary to follow mathematical conventions not just in display equations, but also in the body of the text, for example by making variable names like x italic. Although this could be done by surrounding the appropriate parts with .EQ and .EN, the continual repetition of .EQ and .EN is a nuisance. Furthermore, with '−ms', .EQ and .EN imply a displayed equation.

EQN provides a shorthand for short in-line expressions. You can define two characters to mark the left and right ends of an in-line equation, and then type expressions right in the middle of text lines. To set both the left and right characters to dollar signs, for example, add to the beginning of your document the three lines

```
.EQ
delim $$
.EN
```

Having done this, you can then say things like

> Let $alpha sub i$ be the primary variable, and let $beta$ be zero. Then we can show that $x sub 1$ is $>=0$.

This works as you might expect — spaces, newlines, and so on are significant in the text, but not in the equation part itself. Multiple equations can occur in a single input line.

Enough room is left before and after a line that contains in-line expressions that something like $\sum_{i=1}^{n} x_i$ does not interfere with the lines surrounding it.

To turn off the delimiters,

```
.EQ
delim off
.EN
```

Warning: don't use braces, tildes, circumflexes, or double quotes as delimiters — chaos will result.

20. Definitions

EQN provides a facility so you can give a frequently-used string of characters a name, and thereafter just type the name instead of the whole string. For example, if the sequence

```
x sub i sub 1 + y sub i sub 1
```

appears repeatedly throughout a paper, you can save re-typing it each time by defining it like this:

```
define xy 'x sub i sub 1 + y sub i sub 1'
```

This makes *xy* a shorthand for whatever characters occur between the single quotes in the definition. You can use any character instead of quote to mark the ends of the definition, so long as it doesn't appear inside the definition.

Now you can use *xy* like this:

```
.EQ
f(x) = xy ...
.EN
```

and so on. Each occurrence of *xy* will expand into what it was defined as. Be careful to leave spaces or their equivalent around the name when you actually use it, so EQN will be able to identify it as special.

There are several things to watch out for. First, although definitions can use previous definitions, as in

```
.EQ
define xi ' x sub i '
define xi1 ' xi sub 1 '
.EN
```

don't define something in terms of itself A favorite error is to say

```
define X ' roman X '
```

This is a guaranteed disaster, since X *is* now defined in terms of itself. If you say

```
define X ' roman "X" '
```

however, the quotes protect the second X, and everything works fine.

EQN keywords can be redefined. You can make / mean *over* by saying

```
define / ' over '
```

or redefine *over* as / with

```
define over ' / '
```

If you need different things to print on a terminal and on the typesetter, it is sometimes worth defining a symbol differently in NEQN and EQN. This can be done with *ndefine* and *tdefine*. A definition made with *ndefine* only takes effect if you are running NEQN; if you use *tdefine*, the definition only applies for EQN. Names defined with plain *define* apply to both EQN and NEQN.

21. Local Motions

Although EQN tries to get most things at the right place on the paper, it isn't perfect, and occasionally you will need to tune the output to make it just right. Small extra horizontal spaces can be obtained with tilde and circumflex. You can also say *back n* and *fwd n* to move small amounts horizontally. *n* is how far to move in 1/100's of an em (an em is about the width of the letter 'm'.) Thus *back 50* moves back about half the width of an m. Similarly you can move things up or down with *up n* and *down n*. As with *sub* or *sup*, the local motions affect the next thing in the input, and this can be something arbitrarily complicated if it is enclosed in braces.

22. A Large Example

Here is the complete source for the three display equations in the abstract of this guide.

```
.EQ I
G(z)~mark =~ e sup { ln ~ G(z) }
~=~ exp left (
sum from k>=1 {S sub k z sup k} over k right )
~=~  prod from k>=1 e sup {S sub k z sup k /k}
.EN
.EQ I
lineup = left ( 1 + S sub 1 z +
{ S sub 1 sup 2 z sup 2 } over 2! + ... right )
left ( 1+ { S sub 2 z sup 2 } over 2
+ { S sub 2 sup 2 z sup 4 } over { 2 sup 2 cdot 2! }
+ ... right ) ...
.EN
.EQ I
lineup =  sum from m>=0 left (
sum from
pile { k sub 1 ,k sub 2 ,..., k sub m  >=0
above
k sub 1 +2k sub 2 + ... +mk sub m =m}
{ S sub 1 sup {k sub 1} } over {1 sup k sub 1 k sub 1 ! } ~
{ S sub 2 sup {k sub 2} } over {2 sup k sub 2 k sub 2 ! } ~
...
{ S sub m sup {k sub m} } over {m sup k sub m k sub m ! }
right ) z sup m
.EN
```

23. Keywords, Precedences, Etc.

If you don't use braces, EQN will do operations in the order shown in this list.

dyad vec under bar tilde hat dot dotdot
fwd back down up
fat roman italic bold size
sub sup sqrt over
from to

These operations group to the left:

over sqrt left right

All others group to the right.

Digits, parentheses, brackets, punctuation marks, and these mathematical words are converted to Roman font when encountered:

> sin cos tan sinh cosh tanh arc
> max min lim log ln exp
> Re Im and if for det

These character sequences are recognized and translated as shown.

>=	$\geqslant$
<=	$\leqslant$
==	$\equiv$
!=	$\neq$
+-	$\pm$
->	$\rightarrow$
<-	$\leftarrow$
<<	$\ll$
>>	$\gg$
inf	∞
partial	∂
half	$\frac{1}{2}$
prime	$'$
approx	$\approx$
nothing	
cdot	$\cdot$
times	$\times$
del	∇
grad	∇
...	$\cdots$
,...,	$,\ldots,$
sum	$\sum$
int	$\int$
prod	$\prod$
union	$\bigcup$
inter	$\bigcap$

To obtain Greek letters, simply spell them out in whatever case you want:

DELTA	Δ	iota	ι
GAMMA	Γ	kappa	κ
LAMBDA	Λ	lambda	λ
OMEGA	Ω	mu	μ
PHI	Φ	nu	ν
PI	Π	omega	ω
PSI	Ψ	omicron	o
SIGMA	Σ	phi	ϕ
THETA	Θ	pi	π
UPSILON	Υ	psi	ψ
XI	Ξ	rho	ρ
alpha	α	sigma	σ
beta	β	tau	τ
chi	χ	theta	θ
delta	δ	upsilon	υ
epsilon	ϵ	xi	ξ
eta	η	zeta	ζ
gamma	γ		

These are all the words known to EQN (except for characters with names), together with the section where they are discussed.

above	17, 18	lpile	17
back	21	mark	15
bar	13	matrix	18
bold	12	ndefine	20
ccol	18	over	9
col	18	pile	17
cpile	17	rcol	18
define	20	right	16
delim	19	roman	12
dot	13	rpile	17
dotdot	13	size	12
down	21	sqrt	10
dyad	13	sub	7
fat	12	sup	7
font	12	tdefine	20
from	11	tilde	13
fwd	21	to	11
gfont	12	under	13
gsize	12	up	21
hat	13	vec	13
italic	12	~, ^	4, 6
lcol	18	{ }	8
left	16	"..."	8, 14
lineup	15		

24. Troubleshooting

If you make a mistake in an equation, like leaving out a brace (very common) or having one too many (very common) or having a *sup* with nothing before it (common), EQN will tell you with the message

syntax error between lines x and y, file z

where *x* and *y* are approximately the lines between which the trouble occurred, and *z* is the name of the file in question. The line numbers are approximate — look nearby as well. There are also self-explanatory messages that arise if you leave out a quote or try to run EQN on a non-existent file.

If you want to check a document before actually printing it (on UNIX only),

```
eqn files >/dev/null
```

will throw away the output but print the messages.

If you use something like dollar signs as delimiters, it is easy to leave one out. This causes very strange troubles. The program *checkeq* (on GCOS, use *./checkeq* instead) checks for misplaced or missing dollar signs and similar troubles.

In-line equations can only be so big because of an internal buffer in TROFF. If you get a message "word overflow", you have exceeded this limit. If you print the equation as a displayed equation this message will usually go away. The message "line overflow" indicates you have exceeded an even bigger buffer. The only cure for this is to break the equation into two separate ones.

On a related topic, EQN does not break equations by itself — you must split long equations up across multiple lines by yourself, marking each by a separate .EQ ... .EN sequence. EQN does warn about equations that are too long to fit on one line.

25. Use on UNIX

To print a document that contains mathematics on the UNIX typesetter,

```
eqn files | troff
```

If there are any TROFF options, they go after the TROFF part of the command. For example,

```
eqn files | troff -ms
```

To run the same document on the GCOS typesetter, use

```
eqn files | troff -g (other options) | gcat
```

A compatible version of EQN can be used on devices like teletypes and DASI and GSI terminals which have half-line forward and reverse capabilities. To print equations on a Model 37 teletype, for example, use

```
neqn files | nroff
```

The language for equations recognized by NEQN is identical to that of EQN, although of course the output is more restricted.

To use a GSI or DASI terminal as the output device,

```
neqn files | nroff -Tx
```

where *x* is the terminal type you are using, such as *300* or *300S*.

EQN and NEQN can be used with the TBL program[2] for setting tables that contain mathematics. Use TBL before [N]EQN, like this:

```
tbl files | eqn | troff
tbl files | neqn | nroff
```

26. Acknowledgments

We are deeply indebted to J. F. Ossanna, the author of TROFF, for his willingness to extend TROFF to make our task easier, and for his continuous assistance during the development and evolution of EQN. We are also grateful to A. V. Aho for advice on language design, to S. C. Johnson for assistance with the YACC compiler-compiler, and to all the EQN users who have made helpful suggestions and criticisms.

References

[1] J. F. Ossanna, "NROFF/TROFF User's Manual", Bell Laboratories Computing Science Technical Report #54, 1976.

[2] M. E. Lesk, "Typing Documents on UNIX", Bell Laboratories, 1976.

[3] M. E. Lesk, "TBL — A Program for Setting Tables", Bell Laboratories Computing Science Technical Report #49, 1976.

Section 8

TBL—A PROGRAM TO FORMAT TABLES

INTRODUCTION

tbl, a program to format tables, was developed at Bell Laboratories and is licensed by Western Electric for use on the 8560. The remainder of this section is a reprint of an article describing *tbl*. The Technical Notes section of this manual describes the limitations of this program and any changes made to this program by Tektronix.

Tbl — A Program to Format Tables

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Tbl is a document formatting preprocessor for *troff* or *nroff* which makes even fairly complex tables easy to specify and enter. It is available on the PDP-11 UNIX* system and on Honeywell 6000 GCOS. Tables are made up of columns which may be independently centered, right-adjusted, left-adjusted, or aligned by decimal points. Headings may be placed over single columns or groups of columns. A table entry may contain equations, or may consist of several rows of text. Horizontal or vertical lines may be drawn as desired in the table, and any table or element may be enclosed in a box. For example:

1970 Federal Budget Transfers (in billions of dollars)			
State	Taxes collected	Money spent	Net
New York	22.91	21.35	−1.56
New Jersey	8.33	6.96	−1.37
Connecticut	4.12	3.10	−1.02
Maine	0.74	0.67	−0.07
California	22.29	22.42	+0.13
New Mexico	0.70	1.49	+0.79
Georgia	3.30	4.28	+0.98
Mississippi	1.15	2.32	+1.17
Texas	9.33	11.13	+1.80

January 16, 1979

* UNIX is a Trademark/Service Mark of the Bell System

Tbl — A Program to Format Tables

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

Introduction.

Tbl turns a simple description of a table into a *troff* or *nroff* [1] program (list of commands) that prints the table. *Tbl* may be used on the PDP-11 UNIX [2] system and on the Honeywell 6000 GCOS system. It attempts to isolate a portion of a job that it can successfully handle and leave the remainder for other programs. Thus *tbl* may be used with the equation formatting program *eqn* [3] or various layout macro packages [4,5,6], but does not duplicate their functions.

This memorandum is divided into two parts. First we give the rules for preparing *tbl* input; then some examples are shown. The description of rules is precise but technical, and the beginning user may prefer to read the examples first, as they show some common table arrangements. A section explaining how to invoke *tbl* precedes the examples. To avoid repetition, henceforth read *troff* as *"troff* or *nroff."*

The input to *tbl* is text for a document, with tables preceded by a ".TS" (table start) command and followed by a ".TE" (table end) command. *Tbl* processes the tables, generating *troff* formatting commands, and leaves the remainder of the text unchanged. The ".TS" and ".TE" lines are copied, too, so that *troff* page layout macros (such as the memo formatting macros [4]) can use these lines to delimit and place tables as they see fit. In particular, any arguments on the ".TS" or ".TE" lines are copied but otherwise ignored, and may be used by document layout macro commands.

The format of the input is as follows:

```
text
.TS
table
.TE
text
.TS
table
.TE
text
. . .
```

where the format of each table is as follows:

```
.TS
options ;
format .
data
.TE
```

Each table is independent, and must contain formatting information followed by the data to be entered in the table. The formatting information, which describes the individual columns and rows of the table, may be preceded by a few options that affect the entire table. A detailed description of tables is given in the next section.

Input commands.

As indicated above, a table contains, first, global options, then a format section describing the layout of the table entries, and then the data to be printed. The format and data are always required, but not the options. The various parts of the table are entered as follows:

1) OPTIONS. There may be a single line of options affecting the whole table. If present, this line must follow the .TS line immediately and must contain a list of option names separated by spaces, tabs, or commas, and must be terminated by a semicolon. The allowable options are:

center	— center the table (default is left-adjust);
expand	— make the table as wide as the current line length;
box	— enclose the table in a box;
allbox	— enclose each item in the table in a box;
doublebox	— enclose the table in two boxes;
tab (*x*)	— use *x* instead of tab to separate data items.
linesize (*n*)	— set lines or rules (e.g. from **box**) in *n* point type;
delim (*xy*)	— recognize *x* and *y* as the *eqn* delimiters.

The *tbl* program tries to keep boxed tables on one page by issuing appropriate "need" (.*ne*) commands. These requests are calculated from the number of lines in the tables, and if there are spacing commands embedded in the input, these requests may be inaccurate; use normal *troff* procedures, such as keep-release macros, in that case. The user who must have a multi-page boxed table should use macros designed for this purpose, as explained below under 'Usage.'

2) FORMAT. The format section of the table specifies the layout of the columns. Each line in this section corresponds to one line of the table (except that the last line corresponds to all following lines up to the next .T&, if any — see below), and each line contains a key-letter for each column of the table. It is good practice to separate the key letters for each column by spaces or tabs. Each key-letter is one of the following:

L or **l**	to indicate a left-adjusted column entry;
R or **r**	to indicate a right-adjusted column entry;
C or **c**	to indicate a centered column entry;
N or **n**	to indicate a numerical column entry, to be aligned with other numerical entries so that the units digits of numbers line up;
A or **a**	to indicate an alphabetic subcolumn; all corresponding entries are aligned on the left, and positioned so that the widest is centered within the column (see example on page 12);
S or **s**	to indicate a spanned heading, i.e. to indicate that the entry from the previous column continues across this column (not allowed for the first column, obviously); or
^	to indicate a vertically spanned heading, i.e. to indicate that the entry from the previous row continues down through this row. (Not allowed for the first row of the table, obviously).

When numerical alignment is specified, a location for the decimal point is sought. The rightmost dot (.) adjacent to a digit is used as a decimal point; if there is no dot adjoining a digit, the rightmost digit is used as a units digit; if no alignment is indicated, the item is centered in the column. However, the special non-printing character string \& may be used to override unconditionally dots and digits, or to align alphabetic data; this string lines up where a dot normally would, and then disappears from the final output. In the example below, the items shown at the left will be aligned (in a numerical column) as shown on the right:

```
13              13
4.2              4.2
26.4.12        26.4.12
abc              abc
abc\&           abc
43\&3.22        433.22
749.12         749.12
```

Note: If numerical data are used in the same column with wider **L** or **r** type table entries, the widest *number* is centered relative to the wider **L** or **r** items (**L** is used instead of **l** for readability; they have the same meaning as key-letters). Alignment within the numerical items is preserved. This is similar to the behavior of **a** type data, as explained above. However, alphabetic subcolumns (requested by the **a** key-letter) are always slightly indented relative to **L** items; if necessary, the column width is increased to force this. This is not true for **n** type entries.

Warning: the **n** and **a** items should not be used in the same column.

For readability, the key-letters describing each column should be separated by spaces. The end of the format section is indicated by a period. The layout of the key-letters in the format section resembles the layout of the actual data in the table. Thus a simple format might appear as:

```
c s s
l n n .
```

which specifies a table of three columns. The first line of the table contains a heading centered across all three columns; each remaining line contains a left-adjusted item in the first column followed by two columns of numerical data. A sample table in this format might be:

Overall title		
Item-a	34.22	9.1
Item-b	12.65	.02
Items: c,d,e	23	5.8
Total	69.87	14.92

There are some additional features of the key-letter system:

Horizontal lines — A key-letter may be replaced by '_' (underscore) to indicate a horizontal line in place of the corresponding column entry, or by '=' to indicate a double horizontal line. If an adjacent column contains a horizontal line, or if there are vertical lines adjoining this column, this horizontal line is extended to meet the nearby lines. If any data entry is provided for this column, it is ignored and a warning message is printed.

Vertical lines — A vertical bar may be placed between column key-letters. This will cause a vertical line between the corresponding columns of the table. A vertical bar to the left of the first key-letter or to the right of the last one produces a line at the edge of the table. If two vertical bars appear between key-letters, a double vertical line is drawn.

Space between columns — A number may follow the key-letter. This indicates the amount of separation between this column and the next column. The number normally specifies the separation in *ens* (one en is about the width of the letter 'n').* If the "expand" option is used, then these numbers are multiplied by a constant such that the table is as wide as the current line length. The default column separation

* More precisely, an en is a number of points (1 point = 1/72 inch) equal to half the current type size.

number is 3. If the separation is changed the worst case (largest space requested) governs.

Vertical spanning — Normally, vertically spanned items extending over several rows of the table are centered in their vertical range. If a key-letter is followed by **t** or **T**, any corresponding vertically spanned item will begin at the top line of its range.

Font changes — A key-letter may be followed by a string containing a font name or number preceded by the letter **f** or **F**. This indicates that the corresponding column should be in a different font from the default font (usually Roman). All font names are one or two letters; a one-letter font name should be separated from whatever follows by a space or tab. The single letters **B**, **b**, **I**, and **i** are shorter synonyms for **fB** and **fI**. Font change commands given with the table entries override these specifications.

Point size changes — A key-letter may be followed by the letter **p** or **P** and a number to indicate the point size of the corresponding table entries. The number may be a signed digit, in which case it is taken as an increment or decrement from the current point size. If both a point size and a column separation value are given, one or more blanks must separate them.

Vertical spacing changes — A key-letter may be followed by the letter **v** or **V** and a number to indicate the vertical line spacing to be used within a multi-line corresponding table entry. The number may be a signed digit, in which case it is taken as an increment or decrement from the current vertical spacing. A column separation value must be separated by blanks or some other specification from a vertical spacing request. This request has no effect unless the corresponding table entry is a text block (see below).

Column width indication — A key-letter may be followed by the letter **w** or **W** and a width value in parentheses. This width is used as a minimum column width. If the largest element in the column is not as wide as the width value given after the **w**, the largest element is assumed to be that wide. If the largest element in the column is wider than the specified value, its width is used. The width is also used as a default line length for included text blocks. Normal *troff* units can be used to scale the width value; if none are used, the default is ens. If the width specification is a unitless integer the parentheses may be omitted. If the width value is changed in a column, the *last* one given controls.

Equal width columns — A key-letter may be followed by the letter **e** or **E** to indicate equal width columns. All columns whose key-letters are followed by **e** or **E** are made the same width. This permits the user to get a group of regularly spaced columns.

Note: The order of the above features is immaterial; they need not be separated by spaces, except as indicated above to avoid ambiguities involving point size and font changes. Thus a numerical column entry in italic font and 12 point type with a minimum width of 2.5 inches and separated by 6 ens from the next column could be specified as

```
np12w(2.5i)fI 6
```

Alternative notation — Instead of listing the format of successive lines of a table on consecutive lines of the format section, successive line formats may be given on the same line, separated by commas, so that the format for the example above might have been written:

```
c s s, l n n .
```

Default — Column descriptors missing from the end of a format line are assumed to be **L**. The longest line in the format section, however, defines the number of columns in the table; extra columns in the data are ignored silently.

3) DATA. The data for the table are typed after the format. Normally, each table line is typed as one line of data. Very long input lines can be broken: any line whose last character is \ is combined with the following line (and the \ vanishes). The data for different columns (the table entries) are separated by tabs, or by whatever character has been specified in the option *tabs* option. There are a few special cases:

Troff commands within tables — An input line beginning with a '.' followed by anything but a number is assumed to be a command to *troff* and is passed through unchanged, retaining its position in the table. So, for example, space within a table may be produced by ".sp" commands in the data.

Full width horizontal lines — An input *line* containing only the character _ (underscore) or = (equal sign) is taken to be a single or double line, respectively, extending the full width of the *table.*

Single column horizontal lines — An input table *entry* containing only the character _ or = is taken to be a single or double line extending the full width of the *column.* Such lines are extended to meet horizontal or vertical lines adjoining this column. To obtain these characters explicitly in a column, either precede them by \& or follow them by a space before the usual tab or newline.

Short horizontal lines — An input table *entry* containing only the string _ is taken to be a single line as wide as the contents of the column. It is not extended to meet adjoining lines.

Repeated characters — An input table *entry* containing only a string of the form \R*x* where *x* is any character is replaced by repetitions of the character *x* as wide as the data in the column. The sequence of *x*'s is not extended to meet adjoining columns.

Vertically spanned items — An input table entry containing only the character string \^ indicates that the table entry immediately above spans downward over this row. It is equivalent to a table format key-letter of '^'.

Text blocks — In order to include a block of text as a table entry, precede it by **T{** and follow it by **T}**. Thus the sequence

. . . **T{**
block of
text
T} . . .

is the way to enter, as a single entry in the table, something that cannot conveniently be typed as a simple string between tabs. Note that the **T}** end delimiter must begin a line; additional columns of data may follow after a tab on the same line. See the example on page 10 for an illustration of included text blocks in a table. If more than twenty or thirty text blocks are used in a table, various limits in the *troff* program are likely to be exceeded, producing diagnostics such as 'too many string/macro names' or 'too many number registers.'

Text blocks are pulled out from the table, processed separately by *troff,* and replaced in the table as a solid block. If no line length is specified in the *block of text* itself, or in the table format, the default is to use $L \times C/(N+1)$ where L is the current line length, C is the number of table columns spanned by the text, and N is the total number of columns in the table. The other parameters (point size, font, etc.) used in setting the *block of text* are those in effect at the beginning of the table (including the effect of the ".TS" macro) and any table format specifications of size, spacing and font, using the **p**, **v** and **f** modifiers to the column key-letters. Commands within the text block itself are also recognized, of course. However, *troff* commands within the table data but not within the text block do not affect that block.

Warnings: — Although any number of lines may be present in a table, only the first 200 lines are used in calculating the widths of the various columns. A multi-page table, of course, may be arranged as several single-page tables if this proves to be a problem. Other difficulties with formatting may arise because, in the calculation of column widths all table entries are assumed to be in the font and size being used when the ".TS" command was encountered, except for font and size changes indicated (a) in the table format section and (b) within the table data (as in the entry \s+3\fIdata\fP\s0). Therefore, although arbitrary *troff* requests may be sprinkled in a table, care must be taken to avoid confusing the width calculations; use requests such as '.ps' with care.

4) ADDITIONAL COMMAND LINES. If the format of a table must be changed after many similar lines, as with sub-headings or summarizations, the ".T&" (table continue) command can be used to change column parameters. The outline of such a table input is:

```
.TS
options ;
format .
data
. . .
.T&
format .
data
.T&
format .
data
.TE
```

as in the examples on pages 10 and 12. Using this procedure, each table line can be close to its corresponding format line.

Warning: it is not possible to change the number of columns, the space between columns, the global options such as *box*, or the selection of columns to be made equal width.

Usage.

On UNIX, *tbl* can be run on a simple table with the command

```
tbl input-file | troff
```

but for more complicated use, where there are several input files, and they contain equations and *ms* memorandum layout commands as well as tables, the normal command would be

```
tbl file-1 file-2 . . . | eqn | troff −ms
```

and, of course, the usual options may be used on the *troff* and *eqn* commands. The usage for *nroff* is similar to that for *troff*, but only TELETYPE® Model 37 and Diablo-mechanism (DASI or GSI) terminals can print boxed tables directly.

For the convenience of users employing line printers without adequate driving tables or post-filters, there is a special *−TX* command line option to *tbl* which produces output that does not have fractional line motions in it. The only other command line options recognized by *tbl* are *−ms* and *−mm* which are turned into commands to fetch the corresponding macro files; usually it is more convenient to place these arguments on the *troff* part of the command line, but they are accepted by *tbl* as well.

Note that when *eqn* and *tbl* are used together on the same file *tbl* should be used first. If there are no equations within tables, either order works, but it is usually faster to run *tbl* first, since *eqn* normally produces a larger expansion of the input than *tbl*. However, if there are equations within tables (using the *delim* mechanism in *eqn*), *tbl* must be first or the output will be scrambled. Users must also beware of using equations in **n**-style columns; this is nearly always wrong, since *tbl* attempts to split numerical format items into two parts and this is not possible with equations. The user can defend against this by giving the *delim(xx)* table option; this prevents splitting of numerical columns within the delimiters. For example, if the *eqn* delimiters are *$$*, giving *delim($$)* a numerical column such as "1245 $+- 16$" will be divided after 1245, not after 16.

Tbl limits tables to twenty columns; however, use of more than 16 numerical columns may fail because of limits in *troff*, producing the 'too many number registers' message. *Troff* number registers used by *tbl* must be avoided by the user within tables; these include two-digit names from 31 to 99, and names of the forms #*x*, *x*+, *x*|, ^*x*, and *x*−, where *x* is any lower case letter. The names ##, #−, and #^ are also used in certain circumstances. To conserve number register names, the **n** and **a** formats share a register; hence the restriction above that they may not be used in the same column.

For aid in writing layout macros, *tbl* defines a number register TW which is the table width; it is defined by the time that the ".TE" macro is invoked and may be used in the expansion of that macro. More importantly, to assist in laying out multi-page boxed tables the macro T# is defined to produce the bottom lines and side lines of a boxed table, and then invoked at its end. By use of this macro in the page footer a multi-page table can be boxed. In particular, the *ms* macros can be used to print a multi-page boxed table with a repeated heading by giving the argument H to the ".TS" macro. If the table start macro is written

```
.TS H
```

a line of the form

```
.TH
```

must be given in the table after any table heading (or at the start if none). Material up to the ".TH" is placed at the top of each page of table; the remaining lines in the table are placed on several pages as required. Note that this is *not* a feature of *tbl*, but of the *ms* layout macros.

Examples.

Here are some examples illustrating features of *tbl*. The symbol Ⓣ in the input represents a tab character.

Input:

```
.TS
box;
c c c
l l l.
Language Ⓣ Authors Ⓣ Runs on

Fortran Ⓣ Many Ⓣ Almost anything
PL/1 Ⓣ IBM Ⓣ 360/370
C Ⓣ BTL Ⓣ 11/45,H6000,370
BLISS Ⓣ Carnegie-Mellon Ⓣ PDP-10,11
IDS Ⓣ Honeywell Ⓣ H6000
Pascal Ⓣ Stanford Ⓣ 370
.TE
```

Output:

Language	Authors	Runs on
Fortran	Many	Almost anything
PL/1	IBM	360/370
C	BTL	11/45,H6000,370
BLISS	Carnegie-Mellon	PDP-10,11
IDS	Honeywell	H6000
Pascal	Stanford	370

Input:

```
.TS
allbox;
c s s
c c c
n n n.
AT&T Common Stock
YearⓉPriceⓉDividend
1971Ⓣ41-54Ⓣ$2.60
2Ⓣ41-54Ⓣ2.70
3Ⓣ46-55Ⓣ2.87
4Ⓣ40-53Ⓣ3.24
5Ⓣ45-52Ⓣ3.40
6Ⓣ51-59Ⓣ.95*
.TE
* (first quarter only)
```

Output:

AT&T Common Stock		
Year	Price	Dividend
1971	41-54	$2.60
2	41-54	2.70
3	46-55	2.87
4	40-53	3.24
5	45-52	3.40
6	51-59	.95*

* (first quarter only)

Input:

```
.TS
box;
c s s
c|c|c
l|l|n.
Major New York Bridges
=
BridgeⓉDesignerⓉLength
_
BrooklynⓉJ. A. RoeblingⓉ1595
ManhattanⓉG. LindenthalⓉ1470
WilliamsburgⓉL. L. BuckⓉ1600
_
QueensboroughⓉPalmer &Ⓣ1182
Ⓣ  Hornbostel
_
Ⓣ Ⓣ1380
TriboroughⓉO. H. AmmannⓉ_
Ⓣ Ⓣ383
_
Bronx WhitestoneⓉO. H. AmmannⓉ2300
Throgs NeckⓉO. H. AmmannⓉ1800
_
George WashingtonⓉO. H. AmmannⓉ3500
.TE
```

Output:

Major New York Bridges		
Bridge	Designer	Length
Brooklyn	J. A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380
		383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800
George Washington	O. H. Ammann	3500

Input:

```
.TS
c c
np-2|n|.
ⓣStack
ⓣ_
1ⓣ46
ⓣ_
2ⓣ23
ⓣ_
3ⓣ15
ⓣ_
4ⓣ6.5
ⓣ_
5ⓣ2.1
ⓣ_
.TE
```

Output:

	Stack
1	46
2	23
3	15
4	6.5
5	2.1

Input:

```
.TS
box;
L L L
L L _
L L|LB
L L _
L L L.
januaryⓣfebruaryⓣmarch
aprilⓣmay
juneⓣjulyⓣMonths
augustⓣseptember
octoberⓣnovemberⓣdecember
.TE
```

Output:

january	february	march
april	may	
june	july	**Months**
august	september	
october	november	december

Input:

```
.TS
box;
cfB s s s.
Composition of Foods
_
.T&
c |c s s
c |c s s
c |c |c |c.
Foodⓣ Percent by Weight
\^ⓣ_
\^ⓣProteinⓣFatⓣCarbo-
\^ⓣ\^ⓣ\^ⓣhydrate
_
.T&
l |n |n |n.
Applesⓣ.4ⓣ.5ⓣ13.0
Halibutⓣ18.4ⓣ5.2ⓣ. . .
Lima beansⓣ7.5ⓣ.8ⓣ22.0
Milkⓣ3.3ⓣ4.0ⓣ5.0
Mushroomsⓣ3.5ⓣ.4ⓣ6.0
Rye breadⓣ9.0ⓣ.6ⓣ52.7
.TE
```

Output:

Composition of Foods			
Food	Percent by Weight		
	Protein	Fat	Carbo-hydrate
Apples	.4	.5	13.0
Halibut	18.4	5.2	...
Lima beans	7.5	.8	22.0
Milk	3.3	4.0	5.0
Mushrooms	3.5	.4	6.0
Rye bread	9.0	.6	52.7

Input:

```
.TS
allbox;
cfI  s   s
c   cw(1i)  cw(1i)
lp9 lp9 lp9.
New York Area Rocks
EraⓣFormationⓣAge (years)
PrecambrianⓣReading Prongⓣ>1 billion
PaleozoicⓣManhattan Prongⓣ400 million
MesozoicⓣT{
.na
Newark Basin, incl.
Stockton, Lockatong, and Brunswick
formations; also Watchungs
and Palisades.
T}ⓣ200 million
CenozoicⓣCoastal PlainⓣT{
On Long Island 30,000 years;
Cretaceous sediments redeposited
by recent glaciation.
.ad
T}
.TE
```

Output:

New York Area Rocks		
Era	Formation	Age (years)
Precambrian	Reading Prong	>1 billion
Paleozoic	Manhattan Prong	400 million
Mesozoic	Newark Basin, incl. Stockton, Lockatong, and Brunswick formations; also Watchungs and Palisades.	200 million
Cenozoic	Coastal Plain	On Long Island 30,000 years; Cretaceous sediments redeposited by recent glaciation.

Input:

```
.EQ
delim $$
.EN

. . .

.TS
doublebox;
c c
l l.
Name ⓣ Definition
.sp
.vs +2p
Gamma ⓣ $GAMMA (z) = int sub 0 sup inf  t sup {z-1} e sup -t dt$
Sine ⓣ $sin (x) = 1 over 2i ( e sup ix - e sup -ix )$
Error ⓣ $ roman erf (z) = 2 over sqrt pi int sub 0 sup z e sup {-t sup 2} dt$
Bessel ⓣ $ J sub 0 (z) = 1 over pi int sub 0 sup pi cos ( z sin theta ) d theta $
Zeta ⓣ $ zeta (s) = sum from k=1 to inf k sup -s ~~( Re~s > 1)$
.vs -2p
.TE
```

Output:

Name	Definition
Gamma	$\Gamma(z)=\int_0^\infty t^{z-1}e^{-t}dt$
Sine	$\sin(x)=\frac{1}{2i}(e^{ix}-e^{-ix})$
Error	$\mathrm{erf}(z)=\frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$
Bessel	$J_0(z)=\frac{1}{\pi}\int_0^\pi \cos(z\sin\theta)d\theta$
Zeta	$\zeta(s)=\sum_{k=1}^{\infty} k^{-s}$ (Re $s>1$)

Input:

```
.TS
box, tab(:);
cb s s s s
cp-2 s s s s
c || c | c | c | c
c || c | c | c | c
r2 || n2 | n2 | n2 | n.
Readability of Text
Line Width and Leading for 10-Point Type
=
Line : Set : 1-Point : 2-Point : 4-Point
Width : Solid : Leading : Leading : Leading
_
9 Pica :\-9.3 :\-6.0 :\-5.3 :\-7.1
14 Pica :\-4.5 :\-0.6 :\-0.3 :\-1.7
19 Pica :\-5.0 :\-5.1 : 0.0 :\-2.0
31 Pica :\-3.7 :\-3.8 :\-2.4 :\-3.6
43 Pica :\-9.1 :\-9.0 :\-5.9 :\-8.8
.TE
```

Output:

Readability of Text
Line Width and Leading for 10-Point Type

Line Width	Set Solid	1-Point Leading	2-Point Leading	4-Point Leading
9 Pica	−9.3	−6.0	−5.3	−7.1
14 Pica	−4.5	−0.6	−0.3	−1.7
19 Pica	−5.0	−5.1	0.0	−2.0
31 Pica	−3.7	−3.8	−2.4	−3.6
43 Pica	−9.1	−9.0	−5.9	−8.8

Input:

```
.TS
c s
cip-2 s
l n
a n.
Some London Transport Statistics
(Year 1964)
Railway route milesⓉ244
TubeⓉ66
Sub-surfaceⓉ22
SurfaceⓉ156
.sp .5
.T&
l r
a r.
Passenger traffic \- railway
JourneysⓉ674 million
Average lengthⓉ4.55 miles
Passenger milesⓉ3,066 million
.T&
l r
a r.
Passenger traffic \- road
JourneysⓉ2,252 million
Average lengthⓉ2.26 miles
Passenger milesⓉ5,094 million
.T&
l n
a n.
.sp .5
VehiclesⓉ12,521
Railway motor carsⓉ2,905
Railway trailer carsⓉ1,269
Total railwayⓉ4,174
OmnibusesⓉ8,347
.T&
l n
a n.
.sp .5
StaffⓉ73,739
Administrative, etc.Ⓣ5,582
Civil engineeringⓉ5,134
Electrical eng.Ⓣ1,714
Mech. eng. \- railwayⓉ4,310
Mech. eng. \- roadⓉ9,152
Railway operationsⓉ8,930
Road operationsⓉ35,946
OtherⓉ2,971
.TE
```

Output:

```
    Some London Transport Statistics
               (Year 1964)
Railway route miles            244
  Tube                          66
  Sub-surface                   22
  Surface                      156

Passenger traffic – railway
  Journeys               674 million
  Average length          4.55 miles
  Passenger miles      3,066 million
Passenger traffic – road
  Journeys             2,252 million
  Average length          2.26 miles
  Passenger miles      5,094 million

Vehicles                    12,521
  Railway motor cars         2,905
  Railway trailer cars       1,269
  Total railway              4,174
  Omnibuses                  8,347

Staff                       73,739
  Administrative, etc.       5,582
  Civil engineering          5,134
  Electrical eng.            1,714
  Mech. eng. – railway       4,310
  Mech. eng. – road          9,152
  Railway operations         8,930
  Road operations           35,946
  Other                      2,971
```

Input:

```
.ps 8
.vs 10p
.TS
center box;
c s s
ci s s
c c c
lB l n.
New Jersey Representatives
(Democrats)
.sp .5
NameⓉOffice addressⓉPhone
.sp .5
James J. FlorioⓉ23 S. White Horse Pike, Somerdale 08083Ⓣ609-627-8222
William J. HughesⓉ2920 Atlantic Ave., Atlantic City 08401Ⓣ609-345-4844
James J. HowardⓉ801 Bangs Ave., Asbury Park 07712Ⓣ201-774-1600
Frank Thompson, Jr.Ⓣ10 Rutgers Pl., Trenton 08618Ⓣ609-599-1619
Andrew MaguireⓉ115 W. Passaic St., Rochelle Park 07662Ⓣ201-843-0240
Robert A. RoeⓉU.S.P.O., 194 Ward St., Paterson 07510Ⓣ201-523-5152
Henry HelstoskiⓉ666 Paterson Ave., East Rutherford 07073Ⓣ201-939-9090
Peter W. Rodino, Jr.ⓉSuite 1435A, 970 Broad St., Newark 07102Ⓣ201-645-3213
Joseph G. MinishⓉ308 Main St., Orange 07050Ⓣ201-645-6363
Helen S. MeynerⓉ32 Bridge St., Lambertville 08530Ⓣ609-397-1830
Dominick V. DanielsⓉ895 Bergen Ave., Jersey City 07306Ⓣ201-659-7700
Edward J. PattenⓉNatl. Bank Bldg., Perth Amboy 08861Ⓣ201-826-4610
.sp .5
.T&
ci s s
lB l n.
(Republicans)
.sp .5v
Millicent FenwickⓉ41 N. Bridge St., Somerville 08876Ⓣ201-722-8200
Edwin B. ForsytheⓉ301 Mill St., Moorestown 08057Ⓣ609-235-6622
Matthew J. RinaldoⓉ1961 Morris Ave., Union 07083Ⓣ201-687-4235
.TE
.ps 10
.vs 12p
```

Output:

	New Jersey Representatives *(Democrats)*	
Name	Office address	Phone
James J. Florio	23 S. White Horse Pike, Somerdale 08083	609-627-8222
William J. Hughes	2920 Atlantic Ave., Atlantic City 08401	609-345-4844
James J. Howard	801 Bangs Ave., Asbury Park 07712	201-774-1600
Frank Thompson, Jr.	10 Rutgers Pl., Trenton 08618	609-599-1619
Andrew Maguire	115 W. Passaic St., Rochelle Park 07662	201-843-0240
Robert A. Roe	U.S.P.O., 194 Ward St., Paterson 07510	201-523-5152
Henry Helstoski	666 Paterson Ave., East Rutherford 07073	201-939-9090
Peter W. Rodino, Jr.	Suite 1435A, 970 Broad St., Newark 07102	201-645-3213
Joseph G. Minish	308 Main St., Orange 07050	201-645-6363
Helen S. Meyner	32 Bridge St., Lambertville 08530	609-397-1830
Dominick V. Daniels	895 Bergen Ave., Jersey City 07306	201-659-7700
Edward J. Patten	Natl. Bank Bldg., Perth Amboy 08861	201-826-4610
	(Republicans)	
Millicent Fenwick	41 N. Bridge St., Somerville 08876	201-722-8200
Edwin B. Forsythe	301 Mill St., Moorestown 08057	609-235-6622
Matthew J. Rinaldo	1961 Morris Ave., Union 07083	201-687-4235

This is a paragraph of normal text placed here only to indicate where the left and right margins are. In this way the reader can judge the appearance of centered tables or expanded tables, and observe how such tables are formatted.

Input:

```
.TS
expand;
c s s s
c c c c
l l n n.
Bell Labs Locations
NameⓉAddressⓉArea CodeⓉPhone
HolmdelⓉHolmdel, N. J. 07733Ⓣ201Ⓣ949-3000
Murray HillⓉMurray Hill, N. J. 07974Ⓣ201Ⓣ582-6377
WhippanyⓉWhippany, N. J. 07981Ⓣ201Ⓣ386-3000
Indian HillⓉNaperville, Illinois 60540Ⓣ312Ⓣ690-2000
.TE
```

Output:

	Bell Labs Locations		
Name	Address	Area Code	Phone
Holmdel	Holmdel, N. J. 07733	201	949-3000
Murray Hill	Murray Hill, N. J. 07974	201	582-6377
Whippany	Whippany, N. J. 07981	201	386-3000
Indian Hill	Naperville, Illinois 60540	312	690-2000

Input:

```
.TS
box;
cb  s  s  s
c|c|c  s
ltiw(1i) | ltw(2i) | lp8 | lw(1.6i)p8.
Some Interesting Places

N̄ameⓉDescriptionⓉPractical Information

T̄{
American Museum of Natural History
T}ⓉT{
The collections fill 11.5 acres (Michelin) or 25 acres (MTA)
of exhibition halls on four floors. There is a full-sized replica
of a blue whale and the world's largest star sapphire (stolen in 1964).
T}ⓉHoursⓉ10-5, ex. Sun 11-5, Wed. to 9
\^Ⓣ\^ⓉLocationⓉT{
Central Park West & 79th St.
T}
\^Ⓣ\^ⓉAdmissionⓉDonation: $1.00 asked
\^Ⓣ\^ⓉSubwayⓉAA to 81st St.
\^Ⓣ\^ⓉTelephoneⓉ212-873-4225

B̄ronx ZooⓉT{
About a mile long and .6 mile wide, this is the largest zoo in America.
A lion eats 18 pounds
of meat a day while a sea lion eats 15 pounds of fish.
T}ⓉHoursⓉT{
10-4:30 winter, to 5:00 summer
T}
\^Ⓣ\^ⓉLocationⓉT{
185th St. & Southern Blvd, the Bronx.
T}
\^Ⓣ\^ⓉAdmissionⓉ$1.00, but Tu,We,Th free
\^Ⓣ\^ⓉSubwayⓉ2, 5 to East Tremont Ave.
\^Ⓣ\^ⓉTelephoneⓉ212-933-1759

B̄rooklyn MuseumⓉT{
Five floors of galleries contain American and ancient art.
There are American period rooms and architectural ornaments saved
from wreckers, such as a classical figure from Pennsylvania Station.
T}ⓉHoursⓉWed-Sat, 10-5, Sun 12-5
\^Ⓣ\^ⓉLocationⓉT{
Eastern Parkway & Washington Ave., Brooklyn.
T}
\^Ⓣ\^ⓉAdmissionⓉFree
\^Ⓣ\^ⓉSubwayⓉ2,3 to Eastern Parkway.
\^Ⓣ\^ⓉTelephoneⓉ212-638-5000

T̄{
New-York Historical Society
T}ⓉT{
All the original paintings for Audubon's
.I
Birds of America
.R
are here, as are exhibits of American decorative arts, New York history,
Hudson River school paintings, carriages, and glass paperweights.
T}ⓉHoursⓉT{
Tues-Fri & Sun, 1-5; Sat 10-5
T}
\^Ⓣ\^ⓉLocationⓉT{
Central Park West & 77th St.
T}
\^Ⓣ\^ⓉAdmissionⓉFree
\^Ⓣ\^ⓉSubwayⓉAA to 81st St.
\^Ⓣ\^ⓉTelephoneⓉ212-873-3400
.TE
```

Output:

Some Interesting Places			
Name	Description	Practical Information	
American Museum of Natural History	The collections fill 11.5 acres (Michelin) or 25 acres (MTA) of exhibition halls on four floors. There is a full-sized replica of a blue whale and the world's largest star sapphire (stolen in 1964).	Hours Location Admission Subway Telephone	10-5, ex. Sun 11-5, Wed. to 9 Central Park West & 79th St. Donation: $1.00 asked AA to 81st St. 212-873-4225
Bronx Zoo	About a mile long and .6 mile wide, this is the largest zoo in America. A lion eats 18 pounds of meat a day while a sea lion eats 15 pounds of fish.	Hours Location Admission Subway Telephone	10-4:30 winter, to 5:00 summer 185th St. & Southern Blvd, the Bronx. $1.00, but Tu,We,Th free 2, 5 to East Tremont Ave. 212-933-1759
Brooklyn Museum	Five floors of galleries contain American and ancient art. There are American period rooms and architectural ornaments saved from wreckers, such as a classical figure from Pennsylvania Station.	Hours Location Admission Subway Telephone	Wed-Sat, 10-5, Sun 12-5 Eastern Parkway & Washington Ave., Brooklyn. Free 2,3 to Eastern Parkway. 212-638-5000
New-York Historical Society	All the original paintings for Audubon's *Birds of America* are here, as are exhibits of American decorative arts, New York history, Hudson River school paintings, carriages, and glass paperweights.	Hours Location Admission Subway Telephone	Tues-Fri & Sun, 1-5; Sat 10-5 Central Park West & 77th St. Free AA to 81st St. 212-873-3400

Acknowledgments.

Many thanks are due to J. C. Blinn, who has done a large amount of testing and assisted with the design of the program. He has also written many of the more intelligible sentences in this document and helped edit all of it. All phototypesetting programs on UNIX are dependent on the work of the late J. F. Ossanna, whose assistance with this program in particular had been most helpful. This program is patterned on a table formatter originally written by J. F. Gimpel. The assistance of T. A. Dolotta, B. W. Kernighan, and J. N. Sturman is gratefully acknowledged.

References.

[1] J. F. Ossanna, *NROFF/TROFF User's Manual*, Computing Science Technical Report No. 54, Bell Laboratories, 1976.

[2] K. Thompson and D. M. Ritchie, "The UNIX Time-Sharing System," *Comm. ACM.* **17**, pp. 365–75 (1974).

[3] B. W. Kernighan and L. L. Cherry, "A System for Typesetting Mathematics," *Comm. ACM.* **18**, pp. 151–57 (1975).

[4] M. E. Lesk, *Typing Documents on UNIX*, UNIX Programmer's Manual, Volume 2.

[5] M. E. Lesk and B. W. Kernighan, *Computer Typesetting of Technical Journals on UNIX, Proc. AFIPS NCC*, vol. 46, pp. 879-888 (1977).

[6] J. R. Mashey and D. W. Smith, "Documentation Tools and Techniques," *Proc. 2nd Int. Conf. on Software Engineering*, pp. 177-181 (October, 1976).

List of Tbl Command Characters and Words

Command	*Meaning*	*Section*
a A	Alphabetic subcolumn	2
allbox	Draw box around all items	1
b B	Boldface item	2
box	Draw box around table	1
c C	Centered column	2
center	Center table in page	1
doublebox	Doubled box around table	1
e E	Equal width columns	2
expand	Make table full line width	1
f F	Font change	2
i I	Italic item	2
l L	Left adjusted column	2
n N	Numerical column	2
nnn	Column separation	2
p P	Point size change	2
r R	Right adjusted column	2
s S	Spanned item	2
t T	Vertical spanning at top	2
tab (*x*)	Change data separator character	1
T{ T}	Text block	3
v V	Vertical spacing change	2
w W	Minimum width value	2
.*xx*	Included *troff* command	3
|	Vertical line	2
||	Double vertical line	2
^	Vertical span	2
\^	Vertical span	3
=	Double horizontal line	2,3
_	Horizontal line	2,3
_	Short horizontal line	3
\R*x*	Repeat character	3

Section 9

REFER—SOME APPLICATIONS OF INVERTED INDEXES

INTRODUCTION

refer, a program that uses inverted indexes, was developed at Bell Laboratories and is licensed by Western Electric for use on the 8560. The remainder of this section is a reprint of an article describing *refer*. The Technical Notes section of this manual describes the limitations of this program and any changes made to this program by Tektronix.

Some Applications of Inverted Indexes on the UNIX System

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction.

The UNIX† system has many utilities (e.g. *grep, awk, lex, egrep, fgrep, ...*) to search through files of text, but most of them are based on a linear scan through the entire file, using some deterministic automaton. This memorandum discusses a program which uses inverted indexes[1] and can thus be used on much larger data bases.

As with any indexing system, of course, there are some disadvantages; once an index is made, the files that have been indexed can not be changed without remaking the index. Thus applications are restricted to those making many searches of relatively stable data. Furthermore, these programs depend on hashing, and can only search for exact matches of whole keywords. It is not possible to look for arithmetic or logical expressions (e.g. "date greater than 1970") or for regular expression searching such as that in *lex.*[2]

Currently there are two uses of this software, the *refer* preprocessor to format references, and the *lookall* command to search through all text files on the UNIX system.

The remaining sections of this memorandum discuss the searching programs and their uses. Section 2 explains the operation of the searching algorithm and describes the data collected for use with the *lookall* command. The more important application, *refer* has a user's description in section 3. Section 4 goes into more detail on reference files for the benefit of those who wish to add references to data bases or write new *troff* macros for use with *refer*. The options to make *refer* collect identical citations, or otherwise relocate and adjust references, are described in section 5. The UNIX manual sections for *refer, lookall,* and associated commands are attached as appendices.

2. Searching.

The indexing and searching process is divided into two phases, each made of two parts. These are shown below.

A. Construct the index.

(1) Find keys — turn the input files into a sequence of tags and keys, where each tag identifies a distinct item in the input and the keys for each such item are the strings under which it is to be indexed.

(2) Hash and sort — prepare a set of inverted indexes from which, given a set of keys, the appropriate item tags can be found quickly.

B. Retrieve an item in response to a query.

†UNIX is a Trademark of Bell Laboratories.

1. D. Knuth, *The Art of Computer Programming: Vol. 3, Sorting and Searching,* Addison-Wesley, Reading, Mass. (1977). See section 6.5.

2. M. E. Lesk, "Lex — A Lexical Analyzer Generator," Comp. Sci. Tech. Rep. No. 39, Bell Laboratories, Murray Hill, New Jersey (D).

(3) Search — Given some keys, look through the files prepared by the hashing and sorting facility and derive the appropriate tags.

(4) Deliver — Given the tags, find the original items. This completes the searching process.

The first phase, making the index, is presumably done relatively infrequently. It should, of course, be done whenever the data being indexed change. In contrast, the second phase, retrieving items, is presumably done often, and must be rapid.

An effort is made to separate code which depends on the data being handled from code which depends on the searching procedure. The search algorithm is involved only in steps (2) and (3), while knowledge of the actual data files is needed only by steps (1) and (4). Thus it is easy to adapt to different data files or different search algorithms.

To start with, it is necessary to have some way of selecting or generating keys from input files. For dealing with files that are basically English, we have a key-making program which automatically selects words and passes them to the hashing and sorting program (step 2). The format used has one line for each input item, arranged as follows:

```
name:start,length (tab) key1 key2 key3 ...
```

where *name* is the file name, *start* is the starting byte number, and *length* is the number of bytes in the entry.

These lines are the only input used to make the index. The first field (the file name, byte position, and byte count) is the tag of the item and can be used to retrieve it quickly. Normally, an item is either a whole file or a section of a file delimited by blank lines. After the tab, the second field contains the keys. The keys, if selected by the automatic program, are any alphanumeric strings which are not among the 100 most frequent words in English and which are not entirely numeric (except for four-digit numbers beginning 19, which are accepted as dates). Keys are truncated to six characters and converted to lower case. Some selection is needed if the original items are ver lrge. We normally just take the first *n* keys, with *n* less than 100 or so; this replaces any attempt at intelligent selection. One file in our system is a complete English dictionary; it would presumably be retrieved for all queries.

To generate an inverted index to the list of record tags and keys, the keys are hashed and sorted to produce an index. What is wanted, ideally, is a series of lists showing the tags associated with each key. To condense this, what is actually produced is a list showing the tags associated with each hash code, and thus with some set of keys. To speed up access and further save space, a set of three or possibly four files is produced. These files are:

File	Contents
entry	Pointers to posting file for each hash code
posting	Lists of tag pointers for each hash code
tag	Tags for each item
key	Keys for each item (optional)

The posting file comprises the real data: it contains a sequence of lists of items posted under each hash code. To speed up searching, the entry file is an array of pointers into the posting file, one per potential hash code. Furthermore, the items in the lists in the posting file are not referred to by their complete tag, but just by an address in the tag file, which gives the complete tags. The key file is optional and contains a copy of the keys used in the indexing.

The searching process starts with a query, containing several keys. The goal is to obtain all items which were indexed under these keys. The query keys are hashed, and the pointers in the entry file used to access the lists in the posting file. These lists are addresses in the tag file of documents posted under the hash codes derived from the query. The common items from all lists are determined; this must include the items indexed by every key, but may also contain some items which are false drops, since items referenced by the correct hash codes need not actually have contained the correct keys. Normally, if there are several keys in the query, there are not likely to be many false drops in the final combined list even though each hash code is somewhat ambiguous. The actual tags are then obtained from the tag file, and to guard against the possibility that an item has false-dropped on some hash code in the query, the original items are normally obtained from the delivery program (4) and the query keys checked against them by string comparison.

Usually, therefore, the check for bad drops is made against the original file. However, if the key derivation procedure is complex, it may be preferable to check against the keys fed to program (2). In this case the optional key file which contains the keys associated with each item is generated, and the item tag is supplemented by a string

;start,length

which indicates the starting byte number in the key file and the length of the string of keys for each item. This file is not usually necessary with the present key-selection program, since the keys always appear in the original document.

There is also an option (-C*n*) for coordination level searching. This retrieves items which match all but *n* of the query keys. The items are retrieved in the order of the number of keys that they match. Of course, *n* must be less than the number of query keys (nothing is retrieved unless it matches at least one key).

As an example, consider one set of 4377 references, comprising 660,000 bytes. This included 51,000 keys, of which 5,900 were distinct keys. The hash table is kept full to save space (at the expense of time); 995 of 997 possible hash codes were used. The total set of index files (no key file) included 171,000 bytes, about 26% of the original file size. It took 8 minutes of processor time to hash, sort, and write the index. To search for a single query with the resulting index took 1.9 seconds of processor time, while to find the same paper with a sequential linear search using *grep* (reading all of the tags and keys) took 12.3 seconds of processor time.

We have also used this software to index all of the English stored on our UNIX system. This is the index searched by the *lookall* command. On a typical day there were 29,000 files in our user file system, containing about 152,000,000 bytes. Of these 5,300 files, containing 32,000,000 bytes (about 21%) were English text. The total number of 'words' (determined mechanically) was 5,100,000. Of these 227,000 were selected as keys; 19,000 were distinct, hashing to 4,900 (of 5,000 possible) different hash codes. The resulting inverted file indexes used 845,000 bytes, or about 2.6% of the size of the original files. The particularly small indexes are caused by the fact that keys are taken from only the first 50 non-common words of some very long input files.

Even this large *lookall* index can be searched quickly. For example, to find this document by looking for the keys "lesk inverted indexes" required 1.7 seconds of processor time and system time. By comparison, just to search the 800,000 byte dictionary (smaller than even the inverted indexes, let alone the 32,000,000 bytes of text files) with *grep* takes 29 seconds of processor time. The *lookall* program is thus useful when looking for a document which you believe is stored on-line, but do not know where. For example, many memos from the Computing Science Research Center are in its UNIX file system, but it is often difficult to guess where a particular memo might be (it might have several authors, each with many directories, and have been worked on by a secretary with yet more directories). Instructions for the use of the *lookall* command are given in the manual section, shown in the appendix to this memorandum.

The only indexes maintained routinely are those of publication lists and all English files. To make other indexes, the programs for making keys, sorting them, searching the indexes, and delivering answers must be used. Since they are usually invoked as parts of higher-level commands, they are not in the default command directory, but are available to any user in the directory */usr/lib/refer*. Three programs are of interest: *mkey*, which isolates keys from input files; *inv*, which makes an index from a set of keys; and *hunt*, which searches the index and delivers the items. Note that the two parts of the retrieval phase are combined into one program, to avoid the excessive system work and delay which would result from running these as separate processes.

These three commands have a large number of options to adapt to different kinds of input. The user not interested in the detailed description that now follows may skip to section 3, which describes the *refer* program, a packaged-up version of these tools specifically oriented towards formatting references.

Make Keys. The program *mkey* is the key-making program corresponding to step (1) in phase A. Normally, it reads its input from the file names given as arguments, and if there are no arguments it reads from the standard input. It assumes that blank lines in the input delimit separate items, for each of which a different line of keys should be generated. The lines of keys are written on the standard output. Keys are any alphanumeric string in the input not among the most frequent words in English and not entirely numeric (except that all-numeric strings are acceptable if they are between 1900 and 1999). In the output, keys are translated to lower case, and truncated to six characters in length; any associated punctuation is removed. The following flag arguments are recognized by *mkey:*

−c *name*	Name of file of common words; default is */usr/lib/eign*.
−f *name*	Read a list of files from *name* and take each as an input argument.
−i *chars*	Ignore all lines which begin with '%' followed by any character in *chars*.
−k*n*	Use at most *n* keys per input item.
−l*n*	Ignore items shorter than *n* letters long.
−n*m*	Ignore as a key any word in the first *m* words of the list of common English words. The default is 100.
−s	Remove the labels *(file:start,length)* from the output; just give the keys. Used when searching rather than indexing.
−w	Each whole file is a separate item; blank lines in files are irrelevant.

The normal arguments for indexing references are the defaults, which are *−c /usr/lib/eign*, *−n100*, and *−l3*. For searching, the *−s* option is also needed. When the big *lookall* index of all English files is run, the options are *−w*, *−k50*, and *−f (filelist)*. When running on textual input, the *mkey* program processes about 1000 English words per processor second. Unless the *−k* option is used (and the input files are long enough for it to take effect) the output of *mkey* is comparable in size to its input.

Hash and Invert. The *inv* program computes the hash codes and writes the inverted files. It reads the output of *mkey* and writes the set of files described earlier in this section. It expects one argument, which is used as the base name for the three (or four) files to be written. Assuming an argument of *Index* (the default) the entry file is named *Index.ia*, the posting file *Index.ib*, the tag file *Index.ic*, and the key file (if present) *Index.id*. The *inv* program recognizes the following options:

−a	Append the new keys to a previous set of inverted files, making new files if there is no old set using the same base name.
−d	Write the optional key file. This is needed when you can not check for false drops by looking for the keys in the original inputs, i.e. when the key derivation procedure is complicated and the output keys are not words from the input files.
−h*n*	The hash table size is *n* (default 997); *n* should be prime. Making *n* bigger saves search time and spends disk space.

−i[u] *name*	Take input from file *name*, instead of the standard input; if **u** is present *name* is unlinked when the sort is started. Using this option permits the sort scratch space to overlap the disk space used for input keys.
−n	Make a completely new set of inverted files, ignoring previous files.
−p	Pipe into the sort program, rather than writing a temporary input file. This saves disk space and spends processor time.
−v	Verbose mode; print a summary of the number of keys which finished indexing.

About half the time used in *inv* is in the contained sort. Assuming the sort is roughly linear, however, a guess at the total timing for *inv* is 250 keys per second. The space used is usually of more importance: the entry file uses four bytes per possible hash (note the **−h** option), and the tag file around 15-20 bytes per item indexed. Roughly, the posting file contains one item for each key instance and one item for each possible hash code; the items are two bytes long if the tag file is less than 65336 bytes long, and the items are four bytes wide if the tag file is greater than 65536 bytes long. To minimize storage, the hash tables should be over-full; for most of the files indexed in this way, there is no other real choice, since the *entry* file must fit in memory.

Searching and Retrieving. The *hunt* program retrieves items from an index. It combines, as mentioned above, the two parts of phase (B): search and delivery. The reason why it is efficient to combine delivery and search is partly to avoid starting unnecessary processes, and partly because the delivery operation must be a part of the search operation in any case. Because of the hashing, the search part takes place in two stages: first items are retrieved which have the right hash codes associated with them, and then the actual items are inspected to determine false drops, i.e. to determine if anything with the right hash codes doesn't really have the right keys. Since the original item is retrieved to check on false drops, it is efficient to present it immediately, rather than only giving the tag as output and later retrieving the item again. If there were a separate key file, this argument would not apply, but separate key files are not common.

Input to *hunt* is taken from the standard input, one query per line. Each query should be in *mkey −s* output format; all lower case, no punctuation. The *hunt* program takes one argument which specifies the base name of the index files to be searched. Only one set of index files can be searched at a time, although many text files may be indexed as a group, of course. If one of the text files has been changed since the index, that file is searched with *fgrep;* this may occasionally slow down the searching, and care should be taken to avoid having many out of date files. The following option arguments are recognized by *hunt:*

−a	Give all output; ignore checking for false drops.
−C*n*	Coordination level *n;* retrieve items with not more than *n* terms of the input missing; default *C0*, implying that each search term must be in the output items.
−F[yn*d***]**	"−Fy" gives the text of all the items found; "−Fn" suppresses them. "−F*d*" where *d* is an integer gives the text of the first *d* items. The default is *−Fy.*
−g	Do not use *fgrep* to search files changed since the index was made; print an error comment instead.
−i *string*	Take *string* as input, instead of reading the standard input.
−l *n*	The maximum length of internal lists of candidate items is *n;* default 1000.
−o *string*	Put text output ("−Fy") in *string;* of use *only* when invoked from another program.

−p	Print hash code frequencies; mostly for use in optimizing hash table sizes.
−T[yn*d*]	"−Ty" gives the tags of the items found; "−Tn" suppresses them. "−T*d*" where *d* is an integer gives the first *d* tags. The default is −*Tn*.
−t *string*	Put tag output ("−Ty") in *string;* of use *only* when invoked from another program.

The timing of *hunt* is complex. Normally the hash table is overfull, so that there will be many false drops on any single term; but a multi-term query will have few false drops on all terms. Thus if a query is underspecified (one search term) many potential items will be examined and discarded as false drops, wasting time. If the query is overspecified (a dozen search terms) many keys will be examined only to verify that the single item under consideration has that key posted. The variation of search time with number of keys is shown in the table below. Queries of varying length were constructed to retrieve a particular document from the file of references. In the sequence to the left, search terms were chosen so as to select the desired paper as quickly as possible. In the sequence on the right, terms were chosen inefficiently, so that the query did not uniquely select the desired document until four keys had been used. The same document was the target in each case, and the final set of eight keys are also identical; the differences at five, six and seven keys are produced by measurement error, not by the slightly different key lists.

Efficient Keys				Inefficient Keys			
No. keys	Total drops (incl. false)	Retrieved Documents	Search time (seconds)	No. keys	Total drops (incl. false)	Retrieved Documents	Search time (seconds)
1	15	3	1.27	1	68	55	5.96
2	1	1	0.11	2	29	29	2.72
3	1	1	0.14	3	8	8	0.95
4	1	1	0.17	4	1	1	0.18
5	1	1	0.19	5	1	1	0.21
6	1	1	0.23	6	1	1	0.22
7	1	1	0.27	7	1	1	0.26
8	1	1	0.29	8	1	1	0.29

As would be expected, the optimal search is achieved when the query just specifies the answer; however, overspecification is quite cheap. Roughly, the time required by *hunt* can be approximated as 30 milliseconds per search key plus 75 milliseconds per dropped document (whether it is a false drop or a real answer). In general, overspecification can be recommended; it protects the user against additions to the data base which turn previously uniquely-answered queries into ambiguous queries.

The careful reader will have noted an enormous discrepancy between these times and the earlier quoted time of around 1.9 seconds for a search. The times here are purely for the search and retrieval: they are measured by running many searches through a single invocation of the *hunt* program alone. Usually, the UNIX command processor (the shell) must start both the *mkey* and *hunt* processes for each query, and arrange for the output of *mkey* to be fed to the *hunt* program. This adds a fixed overhead of about 1.7 seconds of processor time to any single search. Furthermore, remember that all these times are processor times: on a typical morning on our PDP 11/70 system, with about one dozen people logged on, to obtain 1 second of processor time for the search program took between 2 and 12 seconds of real time, with a median of 3.9 seconds and a mean of 4.8 seconds. Thus, although the work involved in a single search may be only 200 milliseconds, after you add the 1.7 seconds of startup processor time and then assume a 4:1 elapsed/processor time ratio, it will be 8 seconds before any response is printed.

3. Selecting and Formatting References for TROFF

The major application of the retrieval software is *refer,* which is a *troff* preprocessor like *eqn*.[3] It scans its input looking for items of the form

```
.[
imprecise citation
.]
```

where an imprecise citation is merely a string of words found in the relevant bibliographic citation. This is translated into a properly formatted reference. If the imprecise citation does not correctly identify a single paper (either selecting no papers or too many) a message is given. The data base of citations searched may be tailored to each system, and individual users may specify their own citation files. On our system, the default data base is accumulated from the publication lists of the members of our organization, plus about half a dozen personal bibliographies that were collected. The present total is about 4300 citations, but this increases steadily. Even now, the data base covers a large fraction of local citations.

For example, the reference for the *eqn* paper above was specified as

```
...
preprocessor like
.I eqn.
.[
kernighan cherry acm 1975
.]
It scans its input looking for items
...
```

This paper was itself printed using *refer*. The above input text was processed by *refer* as well as *tbl* and *troff* by the command

```
refer memo-file | tbl | troff −ms
```

and the reference was automatically translated into a correct citation to the ACM paper on mathematical typesetting.

The procedure to use to place a reference in a paper using *refer* is as follows. First, use the *lookbib* command to check that the paper is in the data base and to find out what keys are necessary to retrieve it. This is done by typing *lookbib* and then typing some potential queries until a suitable query is found. For example, had one started to find the *eqn* paper shown above by presenting the query

```
$ lookbib
kernighan cherry
(EOT)
```

lookbib would have found several items; experimentation would quickly have shown that the query given above is adequate. Overspecifying the query is of course harmless; it is even desirable, since it decreases the risk that a document added to the publication data base in the future will be retrieved in addition to the intended document. The extra time taken by even a grossly overspecified query is quite small. A particularly careful reader may have noticed that "acm" does not appear in the printed citation; we have supplemented some of the data base items with extra keywords, such as common abbreviations for journals or other sources, to aid in searching.

If the reference is in the data base, the query that retrieved it can be inserted in the text, between .[and .] brackets. If it is not in the data base, it can be typed into a private file of

3. B. W. Kernighan and L. L. Cherry, "A System for Typesetting Mathematics," *Comm. Assoc. Comp. Mach.* **18**, pp.151-157 (March 1975).

references, using the format discssed in the next section, and then the −p option used to search this private file. Such a command might read (if the private references are called *myfile*)

***refer −p myfile document* | *tbl* | *eqn* | *troff −ms* . . .**

where *tbl* and/or *eqn* could be omitted if not needed. The use of the *−ms* macros[4] or some other macro package, however, is essential. *Refer* only generates the data for the references; exact formatting is done by some macro package, and if none is supplied the references will not be printed.

By default, the references are numbered sequentially, and the *−ms* macros format references as footnotes at the bottom of the page. This memorandum is an example of that style. Other possibilities are discussed in section 5 below.

4. Reference Files.

A reference file is a set of bibliographic references usable with *refer.* It can be indexed using the software described in section 2 for fast searching. What *refer* does is to read the input document stream, looking for imprecise citation references. It then searches through reference files to find the full citations, and inserts them into the document. The format of the full citation is arranged to make it convenient for a macro package, such as the *−ms* macros, to format the reference for printing. Since the format of the final reference is determined by the desired style of output, which is determined by the macros used, *refer* avoids forcing any kind of reference appearance. All it does is define a set of string registers which contain the basic information about the reference; and provide a macro call which is expanded by the macro package to format the reference. It is the responsibility of the final macro package to see that the reference is actually printed; if no macros are used, and the output of *refer* fed untranslated to *troff,* nothing at all will be printed.

The strings defined by *refer* are taken directly from the files of references, which are in the following format. The references should be separated by blank lines. Each reference is a sequence of lines beginning with % and followed by a key-letter. The remainder of that line, and successive lines until the next line beginning with %, contain the information specified by the key-letter. In general, *refer* does not interpret the information, but merely presents it to the macro package for final formatting. A user with a separate macro package, for example, can add new key-letters or use the existing ones for other purposes without bothering *refer.*

The meaning of the key-letters given below, in particular, is that assigned by the *−ms* macros. Not all information, obviously, is used with each citation. For example, if a document is both an internal memorandum and a journal article, the macros ignore the memorandum version and cite only the journal article. Some kinds of information are not used at all in printing the reference; if a user does not like finding references by specifying title or author keywords, and prefers to add specific keywords to the citation, a field is available which is searched but not printed (K).

The key letters currently recognized by *refer* and *−ms,* with the kind of information implied, are:

4. **M. E. Lesk, *Typing Documents on UNIX and GCOS: The -ms Macros for Troff,* Bell Laboratories internal memorandum (1977).**

Key	Information specified	Key	Information specified
A	Author's name	N	Issue number
B	Title of book containing item	O	Other information
C	City of publication	P	Page(s) of article
D	Date	R	Technical report reference
E	Editor of book containing item	T	Title
G	Government (NTIS) ordering number	V	Volume number
I	Issuer (publisher)		
J	Journal name		
K	Keys (for searching)	X	or
L	Label	Y	or
M	Memorandum label	Z	Information not used by *refer*

For example, a sample reference could be typed as:

```
%T Bounds on the Complexity of the Maximal
Common Subsequence Problem
%Z ctr127
%A A. V. Aho
%A D. S. Hirschberg
%A J. D. Ullman
%J J. ACM
%V 23
%N 1
%P 1-12
%M abcd-78
%D Jan. 1976
```

Order is irrelevant, except that authors are shown in the order given. The output of *refer* is a stream of string definitions, one for each of the fields of each reference, as shown below.

```
.]-
.ds [A authors' names ...
.ds [T title ...
.ds [J journal ...
...
.] [ type-number
```

The *refer* program, in general, does not concern itself with the significance of the strings. The different fields are treated identically by *refer*, except that the X, Y and Z fields are ignored (see the **−i** option of *mkey*) in indexing and searching. All *refer* does is select the appropriate citation, based on the keys. The macro package must arrange the strings so as to produce an appropriately formatted citation. In this process, it uses the convention that the 'T' field is the title, the 'J' field the journal, and so forth.

The *refer* program does arrange the citation to simplify the macro package's job, however. The special macro **.]−** precedes the string definitions and the special macro **.] [** follows. These are changed from the input **.[** and **.]** so that running the same file through *refer* again is harmless. The **.]−** macro can be used by the macro package to initialize. The **.] [** macro, which should be used to print the reference, is given an argument *type-number* to indicate the kind of reference, as follows:

Value	Kind of reference
1	Journal article
2	Book
3	Article within book
4	Technical report
5	Bell Labs technical memorandum
0	Other

The type is determined by the presence or absence of particular fields in the citation (a journal article must have a 'J' field, a book must have an 'I' field, and so forth). To a small extent, this violates the above rule that *refer* does not concern itself with the contents of the citation; however, the classification of the citation in *troff* macros would require a relatively expensive and obscure program. Any macro writer may, of course, preserve consistency by ignoring the argument to the **.][** macro.

The reference is flagged in the text with the sequence

```
\*([.number\*(.]
```

where *number* is the footnote number. The strings **[.** and **.]** should be used by the macro package to format the reference flag in the text. These strings can be replaced for a particular footnote, as described in section 5. The footnote number (or other signal) is available to the reference macro **.][** as the string register **[F**. To simplify dealing with a text reference that occurs at the end of a sentence, *refer* treats a reference which follows a period in a special way. The period is removed, and the reference is preceded by a call for the string **<.** and followed by a call for the string **>.** For example, if a reference follows "end." it will appear as

```
end\*(<.\*([.number\*(.]\*(>.
```

where *number* is the footnote number. The macro package should turn either the string **>.** or **<.** into a period and delete the other one. This permits the output to have either the form "end[31]." or "end.^{31}" as the macro package wishes. Note that in one case the period precedes the number and in the other it follows the number.

In some cases users wish to suspend the searching, and merely use the reference macro formatting. That is, the user doesn't want to provide a search key between **.[** and **.]** brackets, but merely the reference lines for the appropriate document. Alternatively, the user can wish to add a few fields to those in the reference as in the standard file, or override some fields. Altering or replacing fields, or supplying whole references, is easily done by inserting lines beginning with **%**; any such line is taken as direct input to the reference processor rather than keys to be searched. Thus

```
.[
key1 key2 key3 ...
%Q New format item
%R Override report name
.]
```

makes the indicates changes to the result of searching for the keys. All of the search keys must be given before the first **%** line.

If no search keys are provided, an entire citation can be provided in-line in the text. For example, if the *eqn* paper citation were to be inserted in this way, rather than by searching for it in the data base, the input would read

```
...
preprocessor like
.I eqn.
.[
%A B. W. Kernighan
%A L. L. Cherry
%T A System for Typesetting Mathematics
%J Comm. ACM
%V 18
%N 3
%P 151-157
%D March 1975
.]
It scans its input looking for items
...
```

This would produce a citation of the same appearance as that resulting from the file search.

As shown, fields are normally turned into *troff* strings. Sometimes users would rather have them defined as macros, so that other *troff* commands can be placed into the data. When this is necessary, simply double the control character **%** in the data. Thus the input

```
.[
%V 23
%%M
Bell Laboratories,
Murray Hill, N.J. 07974
.]
```

is processed by *refer* into

```
.ds [V 23
.de [M
Bell Laboratories,
Murray Hill, N.J. 07974
..
```

The information after **%%M** is defined as a macro to be invoked by **.[M** while the information after **%V** is turned into a string to be invoked by *([V. At present *−ms* expects all information as strings.

5. Collecting References and other Refer Options

Normally, the combination of *refer* and *−ms* formats output as *troff* footnotes which are consecutively numbered and placed at the bottom of the page. However, options exist to place the references at the end; to arrange references alphabetically by senior author; and to indicate references by strings in the text of the form [Name1975a] rather than by number. Whenever references are not placed at the bottom of a page identical references are coalesced.

For example, the **−e** option to *refer* specifies that references are to be collected; in this case they are output whenever the sequence

```
.[
$LIST$
.]
```

is encountered. Thus, to place references at the end of a paper, the user would run *refer* with the *−e* option and place the above **$LIST$** commands after the last line of the text. *Refer* will then move all the references to that point. To aid in formatting the collected references, *refer* writes the references preceded by the line

```
.]<
```

and followed by the line

```
.]>
```

to invoke special macros before and after the references.

Another possible option to *refer* is the **−s** option to specify sorting of references. The default, of course, is to list references in the order presented. The **−s** option implies the **−e** option, and thus requires a

```
.[
$LIST$
.]
```

entry to call out the reference list. The **−s** option may be followed by a string of letters, numbers, and '+' signs indicating how the references are to be sorted. The sort is done using the fields whose key-letters are in the string as sorting keys; the numbers indicate how many of the fields are to be considered, with '+' taken as a large number. Thus the default is **−sAD** meaning "Sort on senior author, then date." To sort on all authors and then title, specify **−sA+T**. And to sort on two authors and then the journal, write **−sA2J**.

Other options to *refer* change the signal or label inserted in the text for each reference. Normally these are just sequential numbers, and their exact placement (within brackets, as superscripts, etc.) is determined by the macro package. The **−l** option replaces reference numbers by strings composed of the senior author's last name, the date, and a disambiguating letter. If a number follows the **l** as in **−l3** only that many letters of the last name are used in the label string. To abbreviate the date as well the form **-l***m,n* shortens the last name to the first *m* letters and the date to the last *n* digits. For example, the option **−l3,2** would refer to the *eqn* paper (reference 3) by the signal *Ker75a*, since it is the first cited reference by Kernighan in 1975.

A user wishing to specify particular labels for a private bibliography may use the **−k** option. Specifying **−k***x* causes the field *x* to be used as a label. The default is **L**. If this field ends in **−**, that character is replaced by a sequence letter; otherwise the field is used exactly as given.

If none of the *refer*-produced signals are desired, the **−b** option entirely suppresses automatic text signals.

If the user wishes to override the *−ms* treatment of the reference signal (which is normally to enclose the number in brackets in *nroff* and make it a superscript in *troff*) this can be done easily. If the lines **.[** or **.]** contain anything following these characters, the remainders of these lines are used to surround the reference signal, instead of the default. Thus, for example, to say "See reference (2)." and avoid "See reference.^{2}" the input might appear

```
See reference
.[ (
imprecise citation ...
.]).
```

Note that blanks are significant in this construction. If a permanent change is desired in the style of reference signals, however, it is probably easier to redefine the strings **[.** and **.]** (which are used to bracket each signal) than to change each citation.

Although normally *refer* limits itself to retrieving the data for the reference, and leaves to a macro package the job of arranging that data as required by the local format, there are two special options for rearrangements that can not be done by macro packages. The **−c** option puts fields into all upper case (CAPS-SMALL CAPS in *troff* output). The key-letters indicated what information is to be translated to upper case follow the **c**, so that **−cAJ** means that authors' names and journals are to be in caps. The **−a** option writes the names of authors last name first, that is *A. D. Hall, Jr.* is written as *Hall, A. D. Jr*. The citation form of the *Journal of the ACM*, for example, would require both **−cA** and **−a** options. This produces authors' names in the style *KERNIGHAN, B. W. AND CHERRY, L. L.* for the previous example. The **−a** option may be followed by a number to indicate how many author names should be reversed; **−a1** (without any **−c** option) would produce *Kernighan, B. W. and L. L. Cherry,* for example.

Finally, there is also the previously-mentioned **−p** option to let the user specify a private file of references to be searched before the public files. Note that *refer* does not insist on a previously made index for these files. If a file is named which contains reference data but is not indexed, it will be searched (more slowly) by *refer* using *fgrep.* In this way it is easy for users to keep small files of new references, which can later be added to the public data bases.

Updating Publication Lists

M. E. Lesk

1. Introduction.

This note describes several commands to update the publication lists. The data base consisting of these lists is kept in a set of files in the directory */usr/dict/papers* on the Version 7 UNIX† system. The reason for having special commands to update these files is that they are indexed, and the only reasonable way to find the items to be updated is to use the index. However, altering the files destroys the usefulness of the index, and makes further editing difficult. So the recommended procedure is to

(1) Prepare additions, deletions, and changes in separate files.

(2) Update the data base and reindex.

Whenever you make changes, etc. it is necessary to run the "add & index" step before logging off; otherwise the changes do not take effect. The next section shows the format of the files in the data base. After that, the procedures for preparing additions, preparing changes, preparing deletions, and updating the public data base are given.

2. Publication Format.

The format of a data base entry is given completely in "Some Applications of Inverted Indexes on UNIX" by M. E. Lesk, the first part of this report, and is summarized here via a few examples. In each example, first the output format for an item is shown, and then the corresponding data base entry.

Journal article:

> A. V. Aho, D. J. Hirschberg, and J. D. Ullman, "Bounds on the Complexity of the Maximal Common Subsequence Problem," *J. Assoc. Comp. Mach.*, vol. 23, no. 1, pp. 1-12 (Jan. 1976).

```
%T Bounds on the Complexity of the Maximal Common
Subsequence Problem
%A A. V. Aho
%A D. S. Hirschberg
%A J. D. Ullman
%J J. Assoc. Comp. Mach.
%V 23
%N 1
%P 1-12
%D Jan. 1976
%M Memo abcd...
```

†UNIX is a Trademark of Bell Laboratories.

Conference proceedings:

B. Prabhala and R. Sethi, "Efficient Computation of Expressions with Common Subexpressions," *Proc. 5th ACM Symp. on Principles of Programming Languages*, pp. 222-230, Tucson, Ariz. (January 1978).

```
%A B. Prabhala
%A R. Sethi
%T Efficient Computation of Expressions with
Common Subexpressions
%J Proc. 5th ACM Symp. on Principles
of Programming Languages
%C Tucson, Ariz.
%D January 1978
%P 222-230
```

Book:

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading, Mass. (1976).

```
%T Software Tools
%A B. W. Kernighan
%A P. J. Plauger
%I Addison-Wesley
%C Reading, Mass.
%D 1976
```

Article within book:

J. W. de Bakker, "Semantics of Programming Languages," pp. 173-227 in *Advances in Information Systems Science, Vol. 2*, ed. J. T. Tou, Plenum Press, New York, N. Y. (1969).

```
%A J. W. de Bakker
%T Semantics of programming languages
%E J. T. Tou
%B Advances in Information Systems Science, Vol. 2
%I Plenum Press
%C New York, N. Y.
%D 1969
%P 173-227
```

Technical Report:

F. E. Allen, "Bibliography on Program Optimization," Report RC-5767, IBM T. J. Watson Research Center, Yorktown Heights, N. Y. (1975).

```
%A F. E. Allen
%D 1975
%T Bibliography on Program Optimization
%R Report RC-5767
%I IBM T. J. Watson Research Center
%C Yorktown Heights, N. Y.
```

Other forms of publication can be entered similarly. Note that conference proceedings are entered as if journals, with the conference name on a *%J* line. This is also sometimes appropriate for obscure publications such as series of lecture notes. When something is both a report and an article, or both a memorandum and an article, enter all necessary information for both; see the first article above, for example. Extra information (such as "In preparation" or "Japanese translation") should be placed on a line beginning *%O*. The most common use of %O lines now is for "Also in ..." to give an additional reference to a secondary appearance of the same paper.

Some of the possible fields of a citation are:

Letter	Meaning	Letter	Meaning
A	Author	K	Extra keys
B	Book including item	N	Issue number
C	City of publication	O	Other
D	Date	P	Page numbers
E	Editor of book	R	Report number
I	Publisher (issuer)	T	Title of item
J	Journal name	V	Volume number

Note that *%B* is used to indicate the title of a book containing the article being entered; when an item is an entire book, the title should be entered with a *%T* as usual.

Normally, the order of items does not matter. The only exception is that if there are multiple authors (%A lines) the order of authors should be that on the paper. If a line is too long, it may be continued on to the next line; any line not beginning with % or . (dot) is assumed to be a continuation of the previous line. Again, see the first article above for an example of a long title. Except for authors, do not repeat any items; if two %J lines are given, for example, the first is ignored. Multiple items on the same file should be separated by blank lines.

Note that in formatted printouts of the file, the exact appearance of the items is determined by a set of macros and the formatting programs. Do not try to adjust fonts, punctuation, etc. by editing the data base; it is wasted effort. In case someone has a real need for a differently-formatted output, a new set of macros can easily be generated to provide alternative appearances of the citations.

3. Updating and Re-indexing.

This section describes the commands that are used to manipulate and change the data base. It explains the procedures for (a) finding references in the data base, (b) adding new references, (c) changing existing references, and (d) deleting references. Remember that all changes, additions, and deletions are done by preparing separate files and then running an 'update and reindex' step.

Checking what's there now. Often you will want to know what is currently in the data base. There is a special command *lookbib* to look for things and print them out. It searches for articles based on words in the title, or the author's name, or the date. For example, you could find the first paper above with

```
lookbib aho ullman maximal subsequence 1976
```

or

```
lookbib aho ullman hirschberg
```

If you don't give enough words, several items will be found; if you spell some wrong, nothing will be found. There are around 4300 papers in the public file; you should always use this command to check when you are not sure whether a certain paper is there or not.

Additions. To add new papers, just type in, on one or more files, the citations for the new papers. Remember to check first if the papers are already in the data base. For example, if a paper has a previous memo version, this should be treated as a change to an existing entry, rather than a new entry. If several new papers are being typed on the same file, be sure that there is a blank line between each two papers.

Changes. To change an item, it should be extracted onto a file. This is done with the command

```
pub.chg key1 key2 key3 ...
```

where the items key1, key2, key3, etc. are a set of keys that will find the paper, as in the *lookbib* command. That is, if

```
lookbib johnson yacc cstr
```

will find a item (to, in this case, Computing Science Technical Report No. 32, "YACC: Yet Another Compiler-Compiler," by S. C. Johnson) then

```
pub.chg johnson yacc cstr
```

will permit you to edit the item. The *pub.chg* command extracts the item onto a file named "bibxxx" where "xxx" is a 3-digit number, e.g. "bib234". The command will print the file name it has chosen. If the set of keys finds more than one paper (or no papers) an error message is printed and no file is written. Each reference to be changed must be extracted with a separate *pub.chg* command, and each will be placed on a separate file. You should then edit the "bibxxx" file as desired to change the item, using the UNIX editor. Do not delete or change the first line of the file, however, which begins %# and is a special code line to tell the update program which item is being altered. You may delete or change other lines, or add lines, as you wish. The changes are not actually made in the public data base until you run the update command *pub.run* (see below). Thus, if after extracting an item and modifying it, you decide that you'd rather leave things as they were, delete the "bibxxx" file, and your change request will disappear.

Deletions. To delete an entry from the data base, type the command

```
pub.del key1 key2 key3 ...
```

where the items key1, key2, etc. are a set of keys that will find the paper, as with the *lookbib* command. That is, if

```
lookbib Aho hirschberg ullman
```

will find a paper,

```
pub.del aho hirschberg ullman
```

deletes it. Note that upper and lower case are equivalent in keys. The *pub.del* command will print the entry being deleted. It also gives the name of a "bibxxx" file on which the deletion command is stored. The actual deletion is not done until the changes, additions, etc. are processed, as with the *pub.chg* command. If, after seeing the item to be deleted, you change your mind about throwing it away, delete the "bibxxx" file and the delete request disappears. Again, if the list of keys does not uniquely identify one paper, an error message is given.

Remember that the default versions of the commands described here edit a public data base. Do not delete items unless you are sure deletion is proper; usually this means that there are duplicate entries for the same paper. Otherwise, view requests for deletion with skepticism; even if one person has no need for a particular item in the data base, someone else may want it there.

If an item is correct, but should not appear in the "List of Publications" as normally produced, add the line

```
%K DNL
```

to the item. This preserves the item intact, but implies "Do Not List" to the to the commands that print publication lists. The DNL line is normally used for some technical reports, minor memoranda, or other low-grade publications.

Update and reindex. When you have completed a session of changes, you should type the command

```
pub.run file1 file2 ...
```

where the names "file1", ... are the new files of additions you have prepared. You need not list the "bibxxx" files representing changes and deletions; they are processed automatically. All of the new items are edited into the standard public data base, and then a new index is made. This process takes about 15 minutes; during this time, searches of the data base will be slower.

Normally, you should execute *pub.run* just before you logoff after performing some edit requests. However, if you don't, the various change request files remain in your directory until you finally do execute *pub.run*. When the changes are processed, the "bibxxx" files are deleted. It is not desirable to wait too long before processing changes, however, to avoid conflicts with someone else who wishes to change the same file. If executing *pub.run* produces the message "File bibxxx too old" it means that someone else has been editing the same file between the time you prepared your changes, and the time you typed *pub.run*. You must delete such old change files and re-enter them.

Note that although *pub.run* discards the "bibxxx" files after processing them, your files of additions are left around even after *pub.run* is finished. If they were typed in only for purposes of updating the data base, you may delete them after they have been processed by *pub.run*.

Example. Suppose, for example, that you wish to

(1) Add to the data base the memos "The Dilogarithm Function of a Real Argument" by R. Morris, and "UNIX Software Distribution by Communication Link," by M. E. Lesk and A. S. Cohen;

(2) Delete from the data base the item "Cheap Typesetters", by M. E. Lesk, SIGLASH Newsletter, 1973; and

(3) Change "J. Assoc. Comp. Mach." to "Jour. ACM" in the citation for Aho, Hirschberg, and Ullman shown above.

The procedure would be as follows. First, you would make a file containing the additions, here called "new.1", in the normal way using the UNIX editor. In the script shown below, the computer prompts are in italics.

```
$ ed new.1
?
a
%T The Dilogarithm Function of a Real Argument
%A Robert Morris
%M abcd
%D 1978

%T UNIX Software Distribution by Communication Link
%A M. E. Lesk
%A A. S. Cohen
%M abcd
%D 1978
w new.1
199
q
```

Next you would specify the deletion, which would be done with the *pub.del* command:

```
        $ pub.del lesk cheap typesetters siglash
```
to which the computer responds:

```
        Will delete: (file bib176)

        %T Cheap Typesetters
        %A M. E. Lesk
        %J ACM SIGLASH Newsletter
        %V 6
        %N 4
        %P 14-16
        %D October 1973
```

And then you would extract the Aho, Hirschberg and Ullman paper. The dialogue involved is shown below. First run *pub.chg* to extract the paper; it responds by printing the citation and informing you that it was placed on file *bib123.* That file is then edited.

```
$ pub.chg aho hirschberg ullman
Extracting as file bib123
%T Bounds on the Complexity of the Maximal
Common Subsequence Problem
%A A. V. Aho
%A D. S. Hirschberg
%A J. D. Ullman
%J J. Assoc. Comp. Mach.
%V 23
%N 1
%P 1-12
%M abcd
%D Jan. 1976

$ ed bib123
312
/Assoc/s/ J/ Jour/p
%J Jour. Assoc. Comp. Mach.
s/Assoc.*/ACM/p
%J Jour. ACM
1,$p
%# /usr/dict/papers/p76 233 245 change
%T Bounds on the Complexity of the Maximal
Common Subsequence Problem
%A A. V. Aho
%A D. S. Hirschberg
%A J. D. Ullman
%J Jour. ACM
%V 23
%N 1
%P 1-12
%M abcd
%D Jan. 1976

w
292
q
$
```

Finally, execute *pub.run*, making sure to remember that you have prepared a new file "new.1":

```
$ pub.run new.1
```

and about fifteen minutes later the new index would be complete and all the changes would be included.

4. Printing a Publication List

There are two commands for printing a publication list, depending on whether you want to print one person's list, or the list of many people. To print a list for one person, use the *pub.indiv* command:

```
pub.indiv M Lesk
```

This runs off the list for M. Lesk and puts it in file "output". Note that no '.' is given after the initial. In case of ambiguity two initials can be used. Similarly, to get the list for group of people, say

```
pub.org xxx
```

which prints all the publications of the members of organization *xxx*, taking the names for the list in the file */usr/dict/papers/centlist/xxx*. This command should normally be run in the background; it takes perhaps 15 minutes. Two options are available with these commands:

```
pub.indiv −p M Lesk
```

prints only the papers, leaving out unpublished notes, patents, etc. Also

```
pub.indiv −t M Lesk | gcat
```

prints a typeset copy, instead of a computer printer copy. In this case it has been directed to an alternate typesetter with the 'gcat' command. These options may be used together, and may be used with the *pub.org* command as well. For example, to print only the papers for all of organization zzz and typeset them, you could type

```
pub.center −t −p zzz | gcat &
```

These publication lists are printed double column with a citation style taken from a set of publication list macros; the macros, of course, can be changed easily to adjust the format of the lists.

MANUAL CHANGE INFORMATION

At Tektronix, we continually strive to keep up with latest electronic developments by adding circuit and component improvements to our instruments as soon as they are developed and tested.

Sometimes, due to printing and shipping requirements, we can't get these changes immediately into printed manuals. Hence, your manual may contain new change information on following pages.

A single change may affect several sections. Since the change information sheets are carried in the manual until all changes are permanently entered, some duplication may occur. If no such change pages appear following this page, your manual is correct as printed.

Tektronix®
COMMITTED TO EXCELLENCE

MANUAL CHANGE INFORMATION

Date: 2-4-82	Change Reference: C1/282
Product: 8560 Text Processing Package Users	Manual Part No.: 070-4272-00

DESCRIPTION

TEXT CORRECTIONS

Page 9-1 At the end of the page, insert the following information:

> The following error message may be generated by the program "refer" if the number of entries in the reference file is too large.
>
> Error: No space for hash list
>
> Use a smaller number of entries.