Writing Papers with GROFF using −me

*Eric P. Allman**

Project INGRES
Electronics Research Laboratory
University of California, Berkeley
Berkeley, California 94720

Modified for GROFF by James Clark

This document describes the text processing facilities available on the UNIX† operating system via GROFF and the −me macro package. It is assumed that the reader already is generally familiar with the UNIX operating system and a text editor such as **ex**. This is intended to be a casual introduction, and as such not all material is covered. In particular, many variations and additional features of the −me macro package are not explained. For a complete discussion of this and other issues, see *The −me Reference Manual* and *The NROFF/TROFF Reference Manual.*

GROFF, a computer program that runs on the UNIX operating system, reads an input file prepared by the user and outputs a formatted paper suitable for publication or framing. The input consists of *text*, or words to be printed, and *requests*, which give instructions to the GROFF program telling how to format the printed copy.

Section 1 describes the basics of text processing. Section 2 describes the basic requests. Section 3 introduces displays. Annotations, such as footnotes, are handled in section 4. The more complex requests which are not discussed in section 2 are covered in section 5. Finally, section 6 discusses things you will need to know if you want to typeset documents. If you are a novice, you probably won't want to read beyond section 4 until you have tried some of the basic features out.

When you have your raw text ready, call the GROFF formatter by typing as a request to the UNIX shell:

groff −me −T*type files*

where *type* describes the type of output device you are using. A complete description of options to the GROFF command can be found in **groff**(1).

The word *argument* is used in this manual to mean a word or number which appears on the same line as a request which modifies the meaning of that request. For example, the request

.sp

spaces one line, but

.sp 4

spaces four lines. The number **4** is an *argument* to the **.sp** request which says to space four lines instead of one. Arguments are separated from the request and from each other by spaces.

---

*Author's current address: Britton Lee, Inc., 1919 Addison Suite 105, Berkeley, California 94704.

†UNIX is a trademark of AT&T Bell Laboratories

1. Basics of Text Processing

The primary function of GROFF is to *collect* words from input lines, *fill* output lines with those words, *justify* the right hand margin by inserting extra spaces in the line, and output the result. For example, the input:

Now is the time
for all good men
to come to the aid
of their party.
Four score and seven
years ago,...

will be read, packed onto output lines, and justified to produce:

Now is the time for all good men to come to the aid of their party. Four score and seven years ago,...

Sometimes you may want to start a new output line even though the line you are on is not yet full; for example, at the end of a paragraph. To do this you can cause a *break*, which starts a new output line. Some requests cause a break automatically, as do blank input lines and input lines beginning with a space.

Not all input lines are text to be formatted. Some of the input lines are *requests* which describe how to format the text. Requests always have a period or an apostrophe ("´") as the first character of the input line.

The text formatter also does more complex things, such as automatically numbering pages, skipping over page folds, putting footnotes in the correct place, and so forth.

I can offer you a few hints for preparing text for input to GROFF. First, keep the input lines short. Short input lines are easier to edit, and GROFF will pack words onto longer lines for you anyhow. In keeping with this, it is helpful to begin a new line after every period, comma, or phrase, since common corrections are to add or delete sentences or phrases. Second, do not put spaces at the end of lines, since this can sometimes confuse the NROFF processor. Third, do not hyphenate words at the end of lines (except words that should have hyphens in them, such as "mother-in-law"); GROFF is smart enough to hyphenate words for you as needed, but is not smart enough to take hyphens out and join a word back together. Also, words such as "mother-in-law" should not be broken over a line, since then you will get a space where not wanted, such as "mother- in-law".

2. Basic Requests

2.1. Paragraphs

Paragraphs are begun by using the **.pp** request. For example, the input:

```
.pp
Now is the time for all good men
to come to the aid of their party.
Four score and seven years ago,...
```

produces a blank line followed by an indented first line. The result is:

  Now is the time for all good men to come to the aid of their party. Four score and seven years ago,...

Notice that the sentences of the paragraphs *must not* begin with a space, since blank lines and lines beginning with spaces cause a break. For example, if I had typed:

```
.pp
Now is the time for all good men
    to come to the aid of their party.
Four score and seven years ago,...
```

The output would be:

Now is the time for all good men
to come to the aid of their party. Four score and seven years ago,...

A new line begins after the word “men” because the second line began with a space character.

There are many fancier types of paragraphs, which will be described later.

2.2. Headers and Footers

Arbitrary headers and footers can be put at the top and bottom of every page. Two requests of the form **.he** *title* and **.fo** *title* define the titles to put at the head and the foot of every page, respectively. The titles are called *three-part* titles, that is, there is a left-justified part, a centered part, and a right-justified part. To separate these three parts the first character of *title* (whatever it may be) is used as a delimiter. Any character may be used, but backslash and double quote marks should be avoided. The percent sign is replaced by the current page number whenever found in the title. For example, the input:

```
.he ´´%´´
.fo ´Jane Jones´´My Book´
```

results in the page number centered at the top of each page, “Jane Jones” in the lower left corner, and “My Book” in the lower right corner.

2.3. Double Spacing

GROFF will double space output text automatically if you use the request **.ls 2**, as is done in this section. You can revert to single spaced mode by typing **.ls 1**.

2.4. Page Layout

A number of requests allow you to change the way the printed copy looks, sometimes called the *layout* of the output page. Most of these requests adjust the placing of “white space” (blank lines or spaces). In these explanations, characters in italics should be replaced with values you wish to use; bold characters represent characters which should actually be typed.

The **.bp** request starts a new page.

The request **.sp** *N* leaves *N* lines of blank space. *N* can be omitted (meaning skip a single line) or can be of the form *N***i** (for *N* inches) or *N***c** (for *N* centimeters). For example, the input:

```
.sp 1.5i
My thoughts on the subject
.sp
```

leaves one and a half inches of space, followed by the line “My thoughts on the subject”, followed by a single blank line.

The **.in** +*N* request changes the amount of white space on the left of the page (the *indent*). The argument *N* can be of the form **+***N* (meaning leave *N* spaces more than you are already leaving), **−***N* (meaning leave less than you do now), or just *N* (meaning leave exactly *N* spaces). *N* can be of the form *N***i** or *N***c** also. For example, the input:

```
initial text
.in 5
some text
.in +1i
more text
.in −2c
final text
```

produces “some text” indented exactly five spaces from the left margin, “more text” indented five spaces plus one inch from the left margin (fifteen spaces on a pica typewriter), and “final text” indented five spaces plus one inch minus two centimeters from the margin. That is, the output is:

```
initial text
          some text
                         more text
             final text
```

The **.ti** +*N* (temporary indent) request is used like **.in** +*N* when the indent should apply to one line only, after which it should revert to the previous indent. For example, the input:

```
.in 1i
.ti 0
Ware, James R.  The Best of Confucius,
Halcyon House, 1950.
An excellent book containing translations of
most of Confucius´ most delightful sayings.
A definite must for anyone interested in the early foundations
of Chinese philosophy.
```

produces:

Ware, James R. The Best of Confucius, Halcyon House, 1950. An excellent book containing translations of most of Confucius' most delightful sayings. A definite must for anyone interested in the early foundations of Chinese philosophy.

Text lines can be centered by using the **.ce** request. The line after the **.ce** is centered (horizontally) on the page. To center more than one line, use **.ce** *N* (where *N* is the number of lines to center), followed by the *N* lines. If you want to center many lines but don't want to count them, type:

```
.ce 1000
lines to center
.ce 0
```

The **.ce 0** request tells GROFF to center zero more lines, in other words, stop centering.

All of these requests cause a break; that is, they always start a new line. If you want to start a new line without performing any other action, use **.br**.

3. Displays

Displays are sections of text to be set off from the body of the paper. Major quotes, tables, and figures are types of displays, as are all the examples used in this document. All displays except centered blocks are output single spaced.

3.1. Major Quotes

Major quotes are quotes which are several lines long, and hence are set in from the rest of the text without quote marks around them. These can be generated using the commands **.(q** and **.)q** to surround the quote. For example, the input:

```
As Weizenbaum points out:
.(q
It is said that to explain is to explain away.
This maxim is nowhere so well fulfilled
as in the areas of computer programming,...
.)q
```

generates as output:

As Weizenbaum points out:

> It is said that to explain is to explain away. This maxim is nowhere so well fulfilled as in the areas of computer programming,...

3.2. Lists

A *list* is an indented, single spaced, unfilled display. Lists should be used when the material to be printed should not be filled and justified like normal text, such as columns of figures or the examples used in this paper. Lists are surrounded by the requests **.(l** and **.)l**. For example, type:

```
Alternatives to avoid deadlock are:
.(l
Lock in a specified order
Detect deadlock and back out one process
Lock all resources needed before proceeding
.)l
```

will produce:
Alternatives to avoid deadlock are:

```
Lock in a specified order
Detect deadlock and back out one process
Lock all resources needed before proceeding
```

3.3. Keeps

A *keep* is a display of lines which are kept on a single page if possible. An example of where you would use a keep might be a diagram. Keeps differ from lists in that lists may be broken over a page boundary whereas keeps will not.

Blocks are the basic kind of keep. They begin with the request **.(b** and end with the request **.)b**. If there is not room on the current page for everything in the block, a new page is begun. This has the unpleasant effect of leaving blank space at the bottom of the page. When this is not appropriate, you can use the alternative, called *floating keeps*.

Floating keeps move relative to the text. Hence, they are good for things which will be referred to by name, such as "See figure 3". A floating keep will appear at the bottom of the current page if it will fit; otherwise, it will appear at the top of the next page. Floating keeps begin with the line **.(z** and end with the line **.)z**. For an example of a floating keep, see figure 1. The **.hl** request is used to draw a horizontal line so that the figure stands out from the text.

```
.(z
.hl
Text of keep to be floated.
.sp
.ce
Figure 1. Example of a Floating Keep.
.hl
.)z
```

Figure 1. Example of a Floating Keep.

3.4. Fancier Displays

Keeps and lists are normally collected in *nofill* mode, so that they are good for tables and such. If you want a display in fill mode (for text), type **.(l F** (Throughout this section, comments applied to **.(l** also apply to **.(b** and **.(z**). This kind of display will be indented from both margins. For example, the input:

```
.(l F
And now boys and girls,
a newer, bigger, better toy than ever before!
Be the first on your block to have your own computer!
Yes kids, you too can have one of these modern
data processing devices.
You too can produce beautifully formatted papers
without even batting an eye!
.)l
```

will be output as:

> And now boys and girls, a newer, bigger, better toy than ever before! Be the first on your block to have your own computer! Yes kids, you too can have one of these modern data processing devices. You too can produce beautifully formatted papers without even batting an eye!

Lists and blocks are also normally indented (floating keeps are normally left justified). To get a left-justified list, type **.(l L**. To get a list centered line-for-line, type **.(l C**. For example, to get a filled, left justified list, enter:

```
.(l L F
text of block
.)l
```

The input:

```
.(l
first line of unfilled display
more lines
.)l
```

produces the indented text:

> first line of unfilled display
> more lines

Typing the character **L** after the **.(l** request produces the left justified result:

first line of unfilled display
more lines

Using **C** instead of **L** produces the line-at-a-time centered output:

first line of unfilled display
more lines

Sometimes it may be that you want to center several lines as a group, rather than centering them one line at a time. To do this use centered blocks, which are surrounded by the requests **.(c** and **.)c**. All the lines are centered as a unit, such that the longest line is centered and the rest are lined up around that line. Notice that lines do not move relative to each other using centered blocks, whereas they do using the **C** argument to keeps.

Centered blocks are *not* keeps, and may be used in conjunction with keeps. For example, to center a group of lines as a unit and keep them on one page, use:

```
.(b L
.(c
first line of unfilled display
more lines
.)c
.)b
```

to produce:

first line of unfilled display
more lines

If the block requests (**.(b** and **.)b**) had been omitted the result would have been the same, but with no guarantee that the lines of the centered block would have all been on one page. Note the use of the **L** argument to **.(b**; this causes the centered block to center within the entire line rather than within the line minus the indent. Also, the center requests must be nested *inside* the keep requests.

4. Annotations

There are a number of requests to save text for later printing. *Footnotes* are printed at the bottom of the current page. *Delayed text* is intended to be a variant form of footnote; the text is printed only when explicitly called for, such as at the end of each chapter. *Indexes* are a type of delayed text having a tag (usually the page number) attached to each entry after a row of dots. Indexes are also saved until called for explicitly.

4.1. Footnotes

Footnotes begin with the request **.(f** and end with the request **.)f**. The current footnote number is maintained automatically, and can be used by typing **, to produce a footnote number[1]. The number is automatically incremented after every footnote. For example, the input:

```
.(q
A man who is not upright
and at the same time is presumptuous;
one who is not diligent and at the same time is ignorant;
one who is untruthful and at the same time is incompetent;
such men I do not count among acquaintances.\**
.(f
\**James R. Ware,
.ul
The Best of Confucius,
Halcyon House, 1950.
Page 77.
.)f
.)q
```

generates the result:

> A man who is not upright and at the same time is presumptuous; one who is not diligent and at the same time is ignorant; one who is untruthful and at the same time is incompetent; such men I do not count among acquaintances.[2]

It is important that the footnote appears *inside* the quote, so that you can be sure that the footnote will appear on the same page as the quote.

---

[1] Like this.

[2] James R. Ware, *The Best of Confucius,* Halcyon House, 1950. Page 77.

4.2. Delayed Text

Delayed text is very similar to a footnote except that it is printed when called for explicitly. This allows a list of references to appear (for example) at the end of each chapter, as is the convention in some disciplines. Use ***#** on delayed text instead of ****** as on footnotes.

If you are using delayed text as your standard reference mechanism, you can still use footnotes, except that you may want to reference them with special characters* rather than numbers.

4.3. Indexes

An "index" (actually more like a table of contents, since the entries are not sorted alphabetically) resembles delayed text, in that it is saved until called for. However, each entry has the page number (or some other tag) appended to the last line of the index entry after a row of dots.

Index entries begin with the request **.(x** and end with **.)x**. The **.)x** request may have a argument, which is the value to print as the "page number". It defaults to the current page number. If the page number given is an underscore ("_") no page number or line of dots is printed at all. To get the line of dots without a page number, type **.)x ""**, which specifies an explicitly null page number.

The **.xp** request prints the index.

For example, the input:

```
.(x
Sealing wax
.)x
.(x
Cabbages and kings
.)x _
.(x
Why the sea is boiling hot
.)x 2.5a
.(x
Whether pigs have wings
.)x ""
.(x
This is a terribly long index entry, such as might be used
for a list of illustrations, tables, or figures; I expect it to
take at least two lines.
.)x
.xp
```

generates:

Sealing wax 8
Cabbages and kings
Why the sea is boiling hot 2.5a
Whether pigs have wings ....
This is a terribly long index entry, such as might be used for a list of illustrations, tables, or figures; I expect it to take at least two lines. 8

The **.(x** request may have a single character argument, specifying the "name" of the index; the normal index is **x**. Thus, several "indices" may be maintained simultaneously (such as a list of tables, table of contents, etc.).

*Such as an asterisk.

Notice that the index must be printed at the *end* of the paper, rather than at the beginning where it will probably appear (as a table of contents); the pages may have to be physically rearranged after printing.

5. Fancier Features

A large number of fancier requests exist, notably requests to provide other sorts of paragraphs, numbered sections of the form **1.2.3** (such as used in this document), and multicolumn output.

5.1. More Paragraphs

Paragraphs generally start with a blank line and with the first line indented. It is possible to get left-justified block-style paragraphs by using **.lp** instead of **.pp**, as demonstrated by the next paragraph.

Sometimes you want to use paragraphs that have the *body* indented, and the first line exdented (opposite of indented) with a label. This can be done with the **.ip** request. A word specified on the same line as **.ip** is printed in the margin, and the body is lined up at a prespecified position (normally five spaces). For example, the input:

```
.ip one
This is the first paragraph.
Notice how the first line
of the resulting paragraph lines up
with the other lines in the paragraph.
.ip two
And here we are at the second paragraph already.
You may notice that the argument to .ip
appears
in the margin.
.lp
We can continue text...
```

produces as output:

one This is the first paragraph. Notice how the first line of the resulting paragraph lines up with the other lines in the paragraph.

two And here we are at the second paragraph already. You may notice that the argument to **.ip** appears in the margin.

We can continue text without starting a new indented paragraph by using the **.lp** request.

If you have spaces in the label of a **.ip** request, you must use an “unpaddable space” instead of a regular space. This is typed as a backslash character (“\”) followed by a space. For example, to print the label “Part 1”, enter:

```
.ip "Part\ 1"
```

If a label of an indented paragraph (that is, the argument to **.ip**) is longer than the space allocated for the label, **.ip** will begin a new line after the label. For example, the input:

```
.ip longlabel
This paragraph had a long label.
The first character of text on the first line
will not line up with the text on second and subsequent lines,
although they will line up with each other.
```

will produce:

longlabel
This paragraph had a long label. The first character of text on the first line will not line up with the text on second and subsequent lines, although they will line up with each other.

It is possible to change the size of the label by using a second argument which is the size of the label. For example, the above example could be done correctly by saying:

```
.ip longlabel 10
```

which will make the paragraph indent 10 spaces for this paragraph only. If you have many paragraphs to indent all the same amount, use the *number register* **ii**. For example, to leave one inch of space for the label, type:

```
.nr ii 1i
```

somewhere before the first call to **.ip**. Refer to the reference manual for more information.

If **.ip** is used with no argument at all no hanging tag will be printed. For example, the input:

```
.ip [a]
This is the first paragraph of the example.
We have seen this sort of example before.
.ip
This paragraph is lined up with the previous paragraph,
but it has no tag in the margin.
```

produces as output:

[a] This is the first paragraph of the example. We have seen this sort of example before.

This paragraph is lined up with the previous paragraph, but it has no tag in the margin.

A special case of **.ip** is **.np**, which automatically numbers paragraphs sequentially from 1. The numbering is reset at the next **.pp**, **.lp**, or **.sh** (to be described in the next section) request. For example, the input:

```
.np
This is the first point.
.np
This is the second point.
Points are just regular paragraphs
which are given sequence numbers automatically
by the .np request.
.pp
This paragraph will reset numbering by .np.
.np
For example,
we have reverted to numbering from one now.
```

generates:

(1) This is the first point.

(2) This is the second point. Points are just regular paragraphs which are given sequence numbers automatically by the .np request.

This paragraph will reset numbering by .np.

(1) For example, we have reverted to numbering from one now.

The **.bu** request gives lists of this sort that are identified with bullets rather than numbers. The paragraphs are also crunched together. For example, the input:

```
.bu
One egg yolk
.bu
One tablespoon cream or top milk
.bu
Salt, cayenne, and lemon juice to taste
.bu
A generous two tablespoonfuls of butter
```

produces[3]:

- One egg yolk
- One tablespoon cream or top milk
- Salt, cayenne, and lemon juice to taste
- A generous two tablespoonfuls of butter

5.2. Section Headings

Section numbers (such as the ones used in this document) can be automatically generated using the **.sh** request. You must tell **.sh** the *depth* of the section number and a section title. The depth specifies how many numbers are to appear (separated by decimal points) in the section number. For example, the section number **4.2.5** has a depth of three.

Section numbers are incremented in a fairly intuitive fashion. If you add a number (increase the depth), the new number starts out at one. If you subtract section numbers (or keep the same number) the final number is incremented. For example, the input:

```
.sh 1 "The Preprocessor"
.sh 2 "Basic Concepts"
.sh 2 "Control Inputs"
.sh 3
.sh 3
.sh 1 "Code Generation"
.sh 3
```

produces as output the result:

1. The Preprocessor
1.1. Basic Concepts
1.2. Control Inputs
1.2.1.
1.2.2.
2. Code Generation
2.1.1.

You can specify the section number to begin by placing the section number after the section title, using spaces instead of dots. For example, the request:

```
.sh 3 "Another section" 7 3 4
```

will begin the section numbered **7.3.4**; all subsequent **.sh** requests will number relative to this number.

There are more complex features which will cause each section to be indented proportionally to the depth of the section. For example, if you enter:

---

[3]By the way, if you put the first three ingredients in a a heavy, deep pan and whisk the ingredients madly over a medium flame (never taking your hand off the handle of the pot) until the mixture reaches the consistency of custard (just a minute or two), then mix in the butter off-heat, you will have a wonderful Hollandaise sauce.

.nr si *N*

each section will be indented by an amount *N*. *N* must have a scaling factor attached, that is, it must be of the form *Nx*, where *x* is a character telling what units *N* is in. Common values for *x* are **i** for inches, **c** for centimeters, and **n** for *ens* (the width of a single character). For example, to indent each section one-half inch, type:

.nr si 0.5i

After this, sections will be indented by one-half inch per level of depth in the section number. For example, this document was produced using the request

.nr si 3n

at the beginning of the input file, giving three spaces of indent per section depth.

Section headers without automatically generated numbers can be done using:

.uh "Title"

which will do a section heading, but will put no number on the section.

5.3. Parts of the Basic Paper

There are some requests which assist in setting up papers. The **.tp** request initializes for a title page. There are no headers or footers on a title page, and unlike other pages you can space down and leave blank space at the top. For example, a typical title page might appear as:

```
.tp
.sp 2i
.(l C
THE GROWTH OF TOENAILS
IN UPPER PRIMATES
.sp
by
.sp
Frank N. Furter
.)l
.bp
```

The **.+c** *T* request can be used to start chapters. Each chapter is automatically numbered from one, and a heading is printed at the top of each chapter with the chapter number and the chapter name *T*. For example, to begin a chapter called "Conclusions", use the request:

.+c "CONCLUSIONS"

which will produce, on a new page, the lines

CHAPTER 5
CONCLUSIONS

with appropriate spacing for a thesis. Also, the header is moved to the foot of the page on the first page of a chapter. Although the **.+c** request was not designed to work only with the **.th** request, it is tuned for the format acceptable for a PhD thesis at Berkeley.

If the title parameter *T* is omitted from the **.+c** request, the result is a chapter with no heading. This can also be used at the beginning of a paper; for example, **.+c** was used to generate page one of this document.

Although papers traditionally have the abstract, table of contents, and so forth at the front of the paper, it is more convenient to format and print them last when using GROFF. This is so that index entries can be collected and then printed for the table of contents (or whatever). At the end of the paper, issue the **.++ P** request, which begins the preliminary part of the paper. After issuing this request, the **.+c** request will begin a preliminary section of the paper. Most notably, this prints the page number restarted from one in lower case Roman numbers. **.+c** may be used repeatedly to begin different parts of the front material for example, the abstract, the table of contents, acknowledgments, list of illustrations, etc. The request **.++ B** may also be used to begin the bibliographic section at the end of the paper. For example, the paper might appear as outlined in figure 2. (In this figure, comments begin with the sequence **\"**.)

```
.th                              \" set for thesis mode
.fo ´´DRAFT´´                    \" define footer for each page
.tp                              \" begin title page
.(l C                            \" center a large block
THE GROWTH OF TOENAILS
IN UPPER PRIMATES
.sp
by
.sp
Frank Furter
.)l                              \" end centered part
.+c INTRODUCTION                 \" begin chapter named "INTRODUCTION"
.(x t                            \" make an entry into index ‘t’
Introduction
.)x                              \" end of index entry
text of chapter one
.+c "NEXT CHAPTER"               \" begin another chapter
.(x t                            \" enter into index ‘t’ again
Next Chapter
.)x
text of chapter two
.+c CONCLUSIONS
.(x t
Conclusions
.)x
text of chapter three
.++ B                            \" begin bibliographic information
.+c BIBLIOGRAPHY                 \" begin another ‘chapter’
.(x t
Bibliography
.)x
text of bibliography
.++ P                            \" begin preliminary material
.+c "TABLE OF CONTENTS"
.xp t                            \" print index ‘t’ collected above
.+c PREFACE                      \" begin another preliminary section
text of preface
```

Figure 2. Outline of a Sample Paper

5.4. Equations and Tables

Two special UNIX programs exist to format special types of material. **Eqn** sets equations. **Tbl** arranges to print extremely pretty tables in a variety of formats. This document will only describe the embellishments to the standard features; consult the reference manuals for those processors for a description of their use.

The **eqn** program is described fully in the document *Typesetting Mathematics – User's Guide* by Brian W. Kernighan and Lorinda L. Cherry. Equations are centered, and are kept on one page. They are introduced by the **.EQ** request and terminated by the **.EN** request.

The **.EQ** request may take an equation number as an optional argument, which is printed vertically centered on the right hand side of the equation. If the equation becomes too long it should be split between two lines. To do this, type:

```
.EQ (eq 34)
text of equation 34
.EN C
.EQ
continuation of equation 34
.EN
```

The **C** on the **.EN** request specifies that the equation will be continued.

The **tbl** program produces tables. It is fully described (including numerous examples) in the document *Tbl – A Program to Format Tables* by M. E. Lesk. Tables begin with the **.TS** request and end with the **.TE** request. Tables are normally kept on a single page. If you have a table which is too big to fit on a single page, so that you know it will extend to several pages, begin the table with the request **.TS H** and put the request **.TH** after the part of the table which you want duplicated at the top of every page that the table is printed on. For example, a table definition for a long table might look like:

```
.TS H
c s s
n n n.
THE TABLE TITLE
.TH
text of the table
.TE
```

5.5. Two Column Output

You can get two column output automatically by using the request **.2c**. This causes everything after it to be output in two-column form. The request **.bc** will start a new column; it differs from **.bp** in that **.bp** may leave a totally blank column when it starts a new page. To revert to single column output, use **.1c**.

5.6. Defining Macros

A *macro* is a collection of requests and text which may be used by stating a simple request. Macros begin with the line **.de** *xx* (where *xx* is the name of the macro to be defined) and end with the line consisting of two dots. After defining the macro, stating the line *.xx* is the same as stating all the other lines. For example, to define a macro that spaces 3 lines and then centers the next input line, enter:

```
.de SS
.sp 3
.ce
..
```

and use it by typing:

```
.SS
Title Line
(beginning of text)
```

Macro names may be one or two characters. In order to avoid conflicts with names in −me, always use upper case letters as names. The only names to avoid are **TS**, **TH**, **TE**, **EQ**, and **EN**.

5.7. Annotations Inside Keeps

Sometimes you may want to put a footnote or index entry inside a keep. For example, if you want to maintain a “list of figures” you will want to do something like:

```
.(z
.(c
text of figure
.)c
.ce
Figure 5.
.(x f
Figure 5
.)x
.)z
```

which you may hope will give you a figure with a label and an entry in the index **f** (presumably a list of figures index). Unfortunately, the index entry is read and interpreted when the keep is read, not when it is printed, so the page number in the index is likely to be wrong. The solution is to use the magic string **\!** at the beginning of all the lines dealing with the index. In other words, you should use:

```
.(z
.(c
Text of figure
.)c
.ce
Figure 5.
\!.(x f
\!Figure 5
\!.)x
.)z
```

which will defer the processing of the index until the figure is output. This will guarantee that the page number in the index is correct. The same comments apply to blocks (with **.(b** and **.)b**) as well.

6. TROFF and the Photosetter

With a little care, you can prepare documents that will print nicely on either a regular terminal or when phototypeset using the TROFF formatting program.

6.1. Fonts

A *font* is a style of type. There are three fonts that are available simultaneously, Times Roman, Times Italic, and Times Bold, plus the special math font. The normal font is Roman.

There are ways of switching between fonts. The requests **.r**, **.i**, **.b**, and **.bi** switch to Roman, italic, bold, and bold-italic fonts respectively. You can set a single word in some font by typing (for example):

```
.i word
```

which will set *word* in italics but does not affect the surrounding text.

Notice that if you are setting more than one word in whatever font, you must surround that word with double quote marks (‘ " ’) so that it will appear to the GROFF processor as a single word. The quote marks will not appear in the formatted text. If you do want a quote mark to appear, you should quote the entire string (even if a single word), and use *two* quote marks where you want one to appear. For example, if you want to produce the text:

"Master Control "

in italics, you must type:

```
.i """Master Control\|"""
```

The **\|** produces a very narrow space so that the “l” does not overlap the quote sign in GROFF, like this:

"Master Control"

There are also some “pseudo-fonts” available. The input:

```
.(b
.u underlined
.bx "words in a box"
.)b
```

generates

underlined
words in a box

Notice that pseudo font requests set only the single parameter in the pseudo font; ordinary font requests will begin setting all text in the special font if you do not provide a parameter. No more than one word should appear with these three font requests in the middle of lines. This is because of the way GROFF justifies text. For example, if you were to issue the requests:

```
.u "some bold italics"
and
.bx "words in a box"
```

in the middle of a line GROFF would produce some bold italics and words in a box, which I think you will agree does not look good.

The second parameter of all font requests is set in the original font. For example, the font request:

```
.b bold face
```

generates “bold” in bold font, but sets “face” in the font of the surrounding text, resulting in:

boldface.

To set the two words **bold** and **face** both in **bold face**, type:

```
.b "bold face"
```

You can mix fonts in a word by using the special sequence **\c** at the end of a line to indicate “continue text processing”; this allows input lines to be joined together without a space between them. For example, the input:

```
.u under \c
.i italics
```

generates under*italics*, but if we had typed:

```
.u under
.i italics
```

the result would have been under *italics* as two words.

6.2. Point Sizes

The phototypesetter supports different sizes of type, measured in points. The default point size is 10 points for most text, 8 points for footnotes. To change the pointsize, type:

.sz +*N*

where *N* is the size wanted in points. The *vertical spacing* (distance between the bottom of most letters (the *baseline*) between adjacent lines) is set to be proportional to the type size.

These pointsize changes are *temporary*!!! For example, to reset the pointsize of basic text to twelve point, use:

```
.nr pp 12
.nr sp 12
.nr tp 12
```

to reset the default pointsize of paragraphs, section headers, and titles respectively. If you only want to set the names of sections in a larger pointsize, use:

.nr sp 11

alone — this sets section titles (e.g., **Point Sizes** above) in a larger font than the default.

A single word or phrase can be set in a smaller pointsize than the surrounding text using the **.sm** request. This is especially convenient for words that are all capitals, due to the optical illusion that makes them look even larger than they actually are. For example:

.sm UNIX

prints as UNIX rather than UNIX.

Warning: changing point sizes on the phototypesetter is a slow mechanical operation. On laser printers it may require loading new fonts. Size changes should be considered carefully.

6.3. Quotes

It is conventional when using the typesetter to use pairs of grave and acute accents to generate double quotes, rather than the double quote character (‘ " ’). This is because it looks better to use grave and acute accents; for example, compare "quote" to ‘‘quote’’.

You may use the sequences ***(lq** and ***(rq** to stand for the left and right quote respectively. For example, use:

```
\*(lqSome things aren´t true
even if they did happen.\*(rq
```

to generate the result:

“Some things aren’t true even if they did happen.”

As a shorthand, the special font request:

.q "quoted text"

will generate “quoted text”. Notice that you must surround the material to be quoted with double quote marks if it is more than one word.

Acknowledgments

I would like to thank Bob Epstein, Bill Joy, and Larry Rowe for having the courage to use the −me macros to produce non-trivial papers during the development stages; Ricki Blau, Pamela Humphrey, and Jim Joyce for their help with the documentation phase; peter kessler for numerous complaints years after I was "done" with this project, most accompanied by fixes (hence forcing me to fix several small bugs); and the plethora of people who have contributed ideas and have given support for the project.

This document applies to the version of the −me macros included with GROFF version 1.20.1.