Advanced Text Processing

Editor's Note: Cross references in the text refer to chapters in the companion book, *UNIX: The Complete Reference, Second Edition,* by Rosen, Host, Klee, Farber, and Rosinski. This chapter contains information that is primarily historical and can help UNIX users understand how text processing was accomplished in the early days of UNIX. While many of the features discussed here have been superceded by newer text processing capabilities such as OpenOffice, many of these features were developed using the basics of **troff** and its preprocessors. In addition, many of these features are available under the current GNU offering of a similar package called **groff**.

In the chapter "Introduction to Text Processing," you learn how to use the **troff** system for basic text processing. You see how to format documents using the memorandum macros together with a few **troff** commands. This is sufficient for the most common text formatting tasks. However, many text formatting tasks cannot be carried out in this way, such as formatting tables, equations, and line drawings. Or you may want to customize your documents with your own particular page layouts and designs. This chapter introduces some advanced UNIX text processing capabilities that you can use to accomplish these and other advanced text formatting tasks.

First, **troff** preprocessors, which you use to produce figures, graphs, tables, and mathematical equations, will be introduced. Next, a survey of selected **troff** commands will be presented that you can use to customize the appearance of documents and to create macros such as those in the **mm** macro package. You will learn how to create your own macros.

You will also learn how to have **troff** switch processing from your source file to a different source file, such as one containing the definitions of macros. You will learn how to create form letters by merging information from separate files.

Finally, you will learn how **troff** documents can take advantage of the PostScript page description language. In particular, you will see how graphics formatted in the PostScript page description language can be inserted into **troff** documents and how to print **troff** documents on PostScript printers. You will also learn how to display documents formatted using **troff** on an X Window terminal.

Preprocessors

It is possible to do just about any typography using **troff**, but it is seldom easy. For example, it is difficult to use **troff** commands directly to format such objects as tables, mathematical equations, pictures, graphs, and so on. You can solve this problem by using a variety of special-purpose programs that operate on a source file, producing output that can be passed on to **troff** for formatting. These special-purpose programs are called **troff** preprocessors because they are used *before* **troff** is run. **troff** preprocessors were developed in accordance with the UNIX philosophy of building tools and little languages to handle special tasks.

When you use a preprocessor, your source file contains instructions for the preprocessor, interspersed with **troff** commands, macro instructions, and text. To produce your output, you have the preprocessor operate on your source file; pipe its output to **troff**; and then pipe the output of **troff** to the typesetter, printer, or display. You can use a sequence of preprocessors, piping the output from each one to the next preprocessor; to **troff**; and then to the typesetter, printer, or display.

This chapter introduces the most commonly used preprocessors:

- **tbl** is used to format tabular material.
- **eqn** is used to format mathematical text.
- **pic** is used to format line drawings.
- **grap** is used to format graphs of various kinds.

Besides these preprocessors, others have been written to format specialized objects such as chemical structures and phonetic symbols.

Formatting Tables

A table is a rectangular arrangement of entries. Formatting tables is a common text formatting task. A versatile **troff** preprocessor for building tables, called **tbl**, was designed by Mike Lesk at Bell Laboratories in 1976. The **tbl** preprocessor makes it possible to produce complicated tables that have an attractive appearance when printed. When you format tables using **tbl**, you include **tbl** instructions and table entries, along with **troff** commands, macros, and text.

The structure of a table follows a general model. A table can be described by specifying global options, such as whether the table should be centered and what should be boxed (that is, enclosed in a box), together with the format for the entries in each row of the table. The instructions you give **tbl** take the following form:

```
.TS
global option line;          [the semicolon is necessary]
row format line 1
row format line 2
.
.
.
last row format line.       [the period is significant]
data for row 1
data for row 2
.
.
.
data for last row
.TE
```

The **.TS/.TE** pair marks the beginning and end of the table. The **tbl** program knows that a table has started once it "sees" the **.TS** instruction, and it knows that the table is completed once it sees the **.TE** instruction. (If you forget to supply the **.TE** instruction, **tbl** treats material beyond the end of your table as part of the table, which produces unintended results.)

The *global option line* describes the overall layout of the table. It consists of a list of global options separated by commas, and it terminates with a semicolon.

The *row format lines* describe how entries are displayed in each row. Each of the initial row format lines describes how one row is displayed. The last row format line describes how all remaining rows are displayed. For example,

```
.TS
center, box, tab(%);
c s s
c | c | c
l | l | n.
Important Mountains of the World
=
Mountain%Location%Altitude (ft)
_
Everest%Nepal-Tibet%29,028.2
Annapurna%Nepal%26,503.1
Nanda Devi%India%25,660.9
Aconcagua%Argentina%22,834.3
McKinley%Alaska%20,299.8
Orizaba%Mexico%18,546.0
Ebert%Colorado%14,431.4
Fuji%Japan%12,394.7
Olympus%Greece%9,730.1
.TE
```

The global options are specified in the second line. Options are separated by commas, and the line of options ends with a semicolon. The options used in this table are these:

- **center** centers the table on the page (the default is left-aligned).
- **box** places a box around the entire table (the default is no box).
- **tab(%)** specifies % as the separator between entries. The default separator is the tab character, but you can specify any character as the separator.

The next three lines specify the format of the rows of the table. The first formatting line specifies the format of the first row of the table, the second formatting line specifies the format of the second row of the table, and the third line specifies the format of *all* remaining rows. The last row formatting line ends with a period.

Row formats are specified by describing the format of the entry in each column. In the example, the *c* in the first row formatting line indicates that the first entry is *centered,* and

Option	Result
center	Center the table on page.
expand	Make the table as wide as current line length.
box	Box the whole table.
doublebox	Box the whole table with double line.
allbox	Enclose each cell in the table with a box.
tab(*x*)	Use the character *x* as data separator.
linesize(*n*)	Set all lines in *n* point type.

TABLE 1 Global Options for tbl

the two *s*'s specify that this entry should also *span* the second and third columns. The *c*'s in the second row formatting line specify that entries in the first, second, and third columns are centered, and the bars (|) specify that entries are separated by vertical lines. Finally, the third row formatting line specifies that in all remaining rows the entries in the first and second columns are *l*eft-aligned, and the entries in the third column are *n*umbers positioned so that their decimal points line up.

After the last row format line, the next line contains the data for the top line of the table. It consists of one entry that spans all three columns. The equal sign (=) in the next line tells **tbl** to insert a double horizontal line across the columns of the table. The next line contains the contents of the next line of the table. It contains entries for the three columns separated by %s. The next line contains an underscore, which tells **tbl** to insert a single horizontal line across the columns of the table. Each line after this contains data for one line of the table.

Global tbl Options

The **tbl** code used to produce the table the previous example uses three global options: **center**, **box**, and **tab(%)**. There are several others that you may want to use. Table 1 summarizes them.

Codes for Laying Out Table Entries

The **tbl** code for the table displayed in the example uses several different codes for laying out elements: **s**, **c**, **l**, and **n**. There are several other codes that you may want to use. These are summarized in Table 2.

You can also specify the font to be used for entries in columns. For instance, the code **lB** produces left-aligned boldface text, and the code **cI** produces centered italic text.

Multiline Entries

Sometimes the entry in one cell of a table requires several lines. To enter several lines of text as one entry, you use a *text block* instruction. The format used to treat blocks of text as single entries (assuming that % is the field separator) is

```
. . .%T{
Block of text
T}%. . .
```

Code	Result
l	Left-align data.
r	Right-align data.
c	Center data.
s	Extend data in previous column to this column.
n	Align numbers by decimal points (or unit places).
a	Indent characters from left alignment by one em space.
t	Vertical span with text on top of column.
^	Expand entry from previous row to this row.

Table 2 Formats for Column Entries in tbl

A text block begins with **T**{ followed by a newline. You enter the text, including any formatting instructions, and conclude the block with newline followed by **T**}. You can then continue entering additional data. The following example illustrates this construction. Here is the **tbl** code:

```
.TS
box, center, tab(%);
cB s
cI | cI
c | l.
troff Preprocessor
_
Preprocessor%Purpose
_
tbl%T{
A preprocessor used to display tabular material.  Entries
are displayed in rows and columns with entries left-justified,
centered, right-justified, and aligned numerically.  Blocks of
text may be used as individual entries.
 T}
_
eqn%T{
A preprocessor used to format mathematical equations.  Equations
can be formatted in displays or can be formatted inline.
Equations can be lined up and matrices can be formatted.
T}
_
pic%T{
A preprocessor used to format pictures.  Basic objects are lines,
arcs, boxes, circles, ellipses, and splines.
T}
.TE
```

Putting Titles on Tables

You can use the **mm** macro **.TB** to number your tables automatically. For instance,

```
.TB "Global Options"
```

produces this title (if this is the seventh time you have used the **.TB** instruction):

Table 7. Global Options

You can place table titles anywhere. Most commonly, table titles are either placed directly before tables or directly after tables. Note that **.TB** is a memorandum macro, and not **tbl** code. This means that you can use this macro even when you do not use **tbl**.

Displaying and Printing When tbl Is Used

You have several ways to produce output when you use **tbl** code. To print the output, you can run **tbl** on the input file, pipe the output to **troff**, and then pipe the output of **troff** to **lp**. So when you use a typesetter or laser printer and have used the **mm** macros, you can print your output by using the following command line:

```
$ tbl file | troff -mm | lp
```

Alternatively, you can use the **mm** or **mmt** commands with the **–t** option, which automatically invokes the table preprocessor. For instance, the command line

```
$ mmt -t file | lp
```

is equivalent to the previous command line. You can display the output on your terminal using

```
$ mm -t file
```

Checking tbl Code

You do not have to produce output to see whether you have inserted **tbl** code correctly and whether there are errors in your code. Some of these errors are identified by the **checkdoc** command. For instance, **checkdoc** checks whether every **.TS** is followed with a **.TE**. However, **checkdoc** will not catch all errors in **tbl** code. To check for possible **tbl** errors, use the following command line:

```
$ tbl file > /dev/null
```

This displays any error messages from **tbl**, discarding the standard output.

Formatting Mathematics

You can format mathematical equations and other mathematical text using the **eqn** preprocessor, designed at Bell Laboratories by Brian Kernighan and Lorinda Cherry in 1975. The **eqn** program includes built-in facilities that let you format mathematical expressions that include arithmetic operations, subscripts and superscripts, fractions, limits, integrals, summations, matrices, Greek letters, and other special mathematical symbols. When you use **eqn**, your source file contains **eqn** code, **troff** commands, macros, and your text. Even if you do not need to do heavy typesetting of equations, you will find **eqn** useful in typesetting commonly used objects such as fractions.

If you use **nroff** rather than **troff**, use the **neqn** preprocessor, instead of **eqn**. **neqn** contains a subset of **eqn** capabilities that work with line printers. **neqn** has many limitations—because it works on line printers, which are being replaced by laser printers, **neqn** is of limited interest.

How to Use eqn

You can use **eqn** in two ways: either to put equations on separate lines or to embed them in text. To format your equations on separate lines, use the **.EQ** and **.EN** instructions, each on its own line, to specify the start and end of the equation, respectively. Insert your **eqn** code for the equation between these instructions.

For example, the following is what you would enter to format an equation:

```
.EQ
x sub 1 ~ = ~ { alpha ~ + ~ pi } over  { beta sup 2 }
.EN
```

- The lines containing **.EQ** and **.EN** mark the beginning and end of the equation, respectively.
- The sub 1 produces a subscript of 1 on x.
- “alpha,” “beta,” and “pi” are typed and will produce these lowercase Greek letters.
- The single brackets { and } group together entries and are not part of the equation itself.
- The word “over” builds a fraction.
- “sup 2” produces a superscript of 2 on “beta.”
- The tildes (~) are used to place blank spaces in the equation. If they are not used, the output will have no space between symbols.

Inline Equations

You can also use **eqn** to place equations within lines of text. To do this, you first specify *delimiters* that are used to mark the beginning and the end of inline equations. You can use almost any delimiters you want, but it is a good idea to use symbols that never, or almost never, occur in your text or equations. Commonly used delimiters include dollar signs ($), number symbols (#), and accents (`). For example, to specify the dollar sign as the delimiter that marks both the beginning and end of an equation, you use the following commands:

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

You can put the previous equation in a line of text as follows:

> After our complicated calculations,
> we find that $x sub 1~=~ {alpha~+~pi} over {beta sup 2}$,
> which is not at all what we expected.

You need to be careful when you use text within inline equations. Blanks inside an inline equation are eliminated by **troff**, so words will run together unless you use tildes (~) between them.

Special Mathematical Symbols

You have seen that **eqn** will produce Greek letters when you spell out the name of the letter. This is the technique **eqn** uses to produce special mathematical symbols that are not ASCII characters. Among the symbols that **eqn** recognizes are the lowercase and uppercase Greek letters, symbols for inequalities, symbols for set operations, the infinity symbol, and so on. Table 3 lists a sampling of the special symbols recognized by **eqn**.

Defining Strings and Symbols for eqn

You can define names that **eqn** will recognize for strings of characters. This is especially useful for defining new symbols. You use the **define**, **tdefine**, or **ndefine** command to define a string for both **eqn** and **neqn**, for **eqn** only, or for **neqn** only. For instance,

```
.EQ
define x1 % x sub 1 %
.EN
```

Input	Output
> =	≥
= =	=
! =	≠
+ −	±
- >	→
inf	∞
prime	′
approx	≈
cdot	·
times	×
grad	∇
int	∫
inter	∩
DELTA	Δ
GAMMA	Γ
XI	Ξ
delta	δ
epsilon	ε
zeta	ξ

TABLE 3 A sampling of special symbols recognized by eqn

makes *x1* the name of the string *x1*. After this definition is made, whenever x1 occurs in an equation, **eqn** translates it to *x1*.

You can also define new symbols using *overstriking*, a **troff** capability discussed later in this chapter. For instance,

```
EQ
define cistar % \o'\(**\(ci'%
.EN
```

makes "cistar" the name of the string "\o'\(**\(ci'". A discussion of the \o escape sequence for overstriking and the escape sequences for special characters used here will be presented in the section "Escape Sequences for Special Effects" later in this chapter.

You can define separate **troff** and **nroff** versions of the same string using the **tdefine** and **ndefine** instructions, which are used analogously to the **define** instruction.

You may find that the symbols you need have already been defined in the public *eqnchar* file, which contains definitions for some frequently used symbols. The *eqnchar* file is located in the */usr/pub* directory. Since symbols produced from these definitions do not print in the same way on different printers, variants of *eqnchar* have been written for different printers. You can find these variants in the */usr/pub* directory as well. Moreover, in DWB 3.*x*, a new file, *posteqnchar,* has been written that optimizes the appearance of symbols for printing with PostScript printers.

If you need to use a special symbol, first see whether it is one of the symbols recognized by **eqn** by checking the full list of these symbols (for instance, in an **eqn** reference manual). If it is not recognized by **eqn**, look in the *eqnchar* file to see whether it is defined there. If it is not there, you can try to define it yourself using such **troff** capabilities as overstriking.

Summations and Related Notations

The **eqn** preprocessor can be used to format equations that contain lower and upper limits as in summation, integrals, and unions. The general construction is

item from *lower limit* to *upper limit*

Examples of this are

```
.EQ
sum from j=1 to inf { 1 over j sup 2 } ~ = ~ {pi sup 2} over 6
.EN
```

and

```
.EQ
int from 0 to { 2 pi } sin (x) dx ~ = ~ 0
.EN
```

This same type of construction also works when only a lower (or only an upper) limit is used, such as for limits. For instance,

```
.EQ
lim from { n -> inf }  a sub n ~ = ~ 0
.EN
```

Lining Up Equations

You can line up equations on several lines using **eqn**. You mark the spot in the first equation that you want to use to line up the rest of the equations with the word "mark." In each remaining equation, you indicate the spot using the word "lineup" where it should be lined up with respect to your designated spot in the first equation. For instance,

```
.EQ
x ~ mark = ~ 10
.EN
.EQ
x sup 2 ~ + ~ y sup 2 ~ lineup = ~ 100
.EN
.EQ
x sup 3 ~ + ~ y sup 3 ~ + ~ z sup 3 ~ lineup = ~ 1000
.EN
```

produces these three lines of equations, with the equal signs aligned.

Formatting Matrices

You can use **eqn** to format a matrix such as a rectangular array of numbers. An example of this is

```
.EQ
left [  pile { 1 above 3 above 2 } ~pile { 0 above 4 above 1 }  right ]
.EN
```

In the preceding code, the "left [" and "right]" are used to produce left and right square brackets, respectively, that are as large as needed to enclose the matrix. The columns are specified using "pile," with the entries in the columns enclosed within { and } and separated with "above."

A Complicated Example

The following complicated example illustrates the versatility of **eqn**. This example was first used in the original guide to using **eqn** and has been used as an example by almost every book that has discussed **eqn**. This book will continue the tradition:

```
.EQ
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
lineup = left (1 + S sub 1 z +
{ S sub 1 sup 2 z sup 2 } over 2
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
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
```

Printing Files Containing eqn Code

To print files containing equations formatted with **eqn**, first run the **eqn** preprocessor on your file, pipe the output to **troff**, and then pipe the output to **lp**:

```
$  eqn file | troff -mm | lp
```

You can also use the **mmt** command with the **–e** option, which arranges for this piping automatically:

```
$ mmt -e file | lp
```

If you use a line printer, use the **neqn** preprocessor and **nroff** to obtain output (providing the limited set of capabilities for formatting equations on line printers),

```
$  neqn file | troff -mm | lp
```

or use the **mm** command with the **–e** option:

```
$ mm -e file | lp
```

If you have used symbols that are defined in the *eqnchar* file, include */usr/pub/eqnchar* before your file on the command line. For example, use

```
$ eqn /usr/pub/eqnchar file | troff -mm | lp
```

as your command line.

To format files containing both tables and equations formatted with **tbl** and **eqn**, use the command line

```
$ tbl file | eqn | troff -mm | lp
```

or equivalently:

```
$ mmt -e -t file | lp
```

Checking eqn Code for Errors

You have two ways to find errors when you use **eqn** before you print or display your output. The first is to use the **checkdoc** command with the name of your file as its argument. (In older releases of the Documenter's Workbench, the **checkdoc** functions that check the format of **eqn** code are included in the **checkeq** command or the **checkmm** command.)

The output of this command will be a list of certain types of **eqn** errors. The second way is to use the command line

```
$  eqn file >/dev/null
```

The standard output of **eqn**, which you do not want to see, is discarded. The standard error, which will display **eqn** errors, will be printed on your screen.

Sometimes you may want to put the list of errors in a separate file. To do this, use the following command:

```
$  eqn file >/dev/null 2>eqnerrors
```

The standard error of the **eqn** command will be put into the file *eqnerrors.*

An inefficient way to find errors made when using **eqn** is to print your document. However, if you have a bitmapped display, you may be able to display your formatted document on your screen. For example, you may do this with **troff** tools for the X Window System. You'll see more about this later in this chapter, in the "Customizing Documents" section.

Formatting Pictures

The **pic** preprocessor, designed in 1981 by Brian Kernighan at Bell Laboratories, provides a language for drawing pictures. (Note that you can produce pictures in many other, more modern ways, and include them in **troff** documents via the inclusion of PostScript files. See later sections of this chapter for details.) When you use **pic**, you describe your picture by specifying the motions used to draw it and the objects you wish to draw.

The format used to insert pictures into **troff** documents is

```
.PS optional-width optional-height
macro definitions
variable assignments
pic code specifying object, motions, and positions
.PE
```

You can use **pic** to draw boxes, circles, ellipses, lines, arrows, arcs, and splines, as well as to insert text. However, **pic** cannot be used to produce artwork, color gradations, or other "high-end" illustrations. It is intended only for simple line drawings, such as flowcharts. You can give **pic** instructions to move either right or left or up or down as you draw your picture. You can place objects relative to other objects in various ways. When this is done, objects are given names so that they can be referred to.

pic Examples

Instead of a lengthy, comprehensive treatment of **pic**, several examples of increasing sophistication will be presented, illustrating how **pic** is used.

A Simple Example There are many ways to specify positions of objects. Here is an example that illustrates how to position objects relative to other objects:

```
.PS
F:  ellipse
ellipse ht .2 wid .3 with .se at F.nw
ellipse ht .2 wid .3 with .sw at F.ne
```

```
circle rad .05 at 0.4 <F.nw,F.c>
circle rad .05 at 0.4 <F.ne,F.c>
arc from 0.3 <F.w,F.e> to 0.7 <F.w,F.e>
.PE
```

In this **pic** code, the first line after the **.PS** draws an ellipse and assigns it the name *F.* The second and third lines draw 0.2 inch by 0.3 inch ellipses with the southeast corner of the first of these at the northwest corner of *F,* and the southwest corner of the second of these at the northeast corner of *F.* The fourth and fifth lines draw circles of radius .05" with centers at points that are 0.4" from the northwest corner of *F* and the center of *F,* and the northeast corner of F and the center of *F,* respectively. Finally, the sixth line draws an arc between the points that are 0.3 and 0.7 of the way along the line segment from the west and east sides of *F,* respectively.

A Second Example

Here is another example illustrating how **pic** code is used. This example shows how *invisible boxes* are used, how **pic** positions objects, and how splines are used:

```
.PS
box invis "input" "file"; arrow
box "pic"; arrow
box "tbl"; arrow
box "eqn"; arrow
box "troff"; arrow
box invis "printer"
[ circle "mm" "macros"; spline right then up -> ] with .ne at 2nd last box.s
.PE
```

The first line of **pic** code in this example produces the words "input" and "file" on two lines inside an *invisible box,* that is, with no box around them. Then an arrow is drawn from the east side of this invisible box.

The next four lines produce boxes with the indicated text inside them, each followed by an arrow. The sixth line produces an invisible box with the word "printer" inside.

The final line produces a circle with the text "mm macros" inside with a spline going from the east side of the circle, and then curving up. This entire object (grouped into one object by the brackets)—the circle, text, and spline—is positioned so that its northeast corner is at the south side of the box containing the word "troff."

Advanced pic Capabilities

You can construct complicated pictures with **pic** by taking advantage of its programmability. For instance, **pic** code can contain conditional statements and loops, as well as certain built-in mathematical functions. The following example illustrates some of these more advanced features of **pic**:

```
.PS
pi=3.1416
r=0.6; dr=0.4
C: circle rad r
circle rad r+dr with .c at C.c
for k=1 to 9
```

```
{
circle rad 0.15 with .c at C.c+(0.8*cos((2*pi*k)/9,0.8*sin((2*pi*k)/9)
}
.PE
```

Here, values are assigned to *pi*, *r*, and *dr*. Then a circle is drawn, which is assigned the name *C*, with radius *r=0.6*. Next, a circle is drawn with radius *r+dr=0.6+0.4=1.0*, with the center the same as the center of the circle *C*, so that this second circle is concentric with *C*. Next, a loop is used to draw nine circles, each with a radius of 0.15 inch, midway between the two concentric circles. This is done by specifying their centers relative to the center of *C* using the built-in trigonometric functions, sine and cosine.

Printing Pictures Formatted with pic

To obtain output from files containing pictures formatted with **pic**, run **pic**, pipe the output to **troff**, and pipe the output of **troff** to **lp**:

```
$ pic file  troff -mm | lp
```

or you can use

```
$ mmt -p file | lp
```

You can format files containing **pic**, **tbl**, or **eqn** instructions with the following command line:

```
$ pic file | tbl | eqn | troff -mm | lp
```

This pipes output from **pic** to **tbl**, which pipes its output to **eqn**, which pipes its output to **troff**, which pipes its output to the printer command. You can also use

```
$ mmt -p -e -t file | lp
```

Front End to pic

Although **pic** is powerful, using it directly can be extremely complicated and tedious. One way to get around this is to use PICASSO, part of DWBX. PICASSO is an interactive program that runs under the OPEN LOOK Graphical User Interface on the X Window System. PICASSO lets you draw directly on a bitmapped display and accepts both mouse and menu commands. For details on using PICASSO see the DWBX 3.4 manual that is part of the DWB Release 3.4 Documentation.

Even if you generate pictures using a graphical front end such as PICASSO, you may still find it worthwhile to understand how to use **pic**, since you can edit the **pic** code produced by one of these front ends. Furthermore, for some applications, direct use of **pic** is more efficient than a graphical front end. Such a situation arises when you use **pic** control structures such as loops to draw a picture with many repeated elements, such as dozens of circles or boxes in a periodic pattern.

Drawing Graphs

You may need to include various types of graphs in your documents. **grap** is a preprocessor to **pic**, which itself is a preprocessor to **troff**, that you can use to specify graphs. It was developed at Bell Laboratories in 1984 by John Bentley and Brian Kernighan. The general format for a graph is

```
.G1
macro definitions
```

```
variable assignments
instructions
.G2
```

You can produce a wide variety of graphs using **grap**. The following example illustrates some of the capabilities of **grap**. For a complete description of **grap** commands, consult the references listed at the end of the chapter.

In this graph, data points are connected by line segments. The comments (set off with \") describe what each instruction does:

```
.G1         \"start graph
label left "Millions of" "People" left .1   \"insert left label .1 inch
   left of graph
label bot "Cases of Swine Flu by Year"      \"insert bottom label
ticks left out from 0 to 60 by 10           \"insert tick marks by 10s
   from 0 to 60 on left
ticks bot out from 1976 to 1983             \"insert bottom tick marks by
   1s from 1976 to 1983
draw solid;                                 \"draw solid lines between
   points
1976 0                                      \"next eight lines give data
   graph
1977 1
1978 4
1979 12
1980 22
1981 43
1982 37
1983 54
.G2
```

Printing Graphs

To print a document containing graphs formatted with **grap**, use the following command line:

```
$ grap file | pic | troff -mm | lp
```

or equivalently:

```
$ mmt -g -p file | lp
```

Customizing Documents

Although the **mm** macros can meet many needs, there are situations when they cannot do what you want. To fine-tune the appearance of your document, you will need to use individual **troff** commands instead of prepackaged macros.

A wide range of **troff** commands are available for customizing your document. These include commands that specify page layout, control the placement of text, produce special symbols, and carry out other specialized tasks.

You can also develop your own macros, which you build by combining **troff** commands, to carry out combinations of tasks you need to use together frequently. Once you have defined a macro, you can use it as an instruction in much the same way that you use an **mm** macro.

In the following discussion, you will learn about important **troff** commands and how to use them to customize your documents. You will also learn how to build your own macros.

An Overview of troff Commands

The collection of **troff** commands can be used to do almost any text formatting task. To have this flexibility and power, a large number of different commands are needed. The chapter "Introduction to Text Processing" introduces several frequently used **troff** commands. However, there are many other **troff** commands that are used only in special situations.

Only the most widely used **troff** commands will be covered, including many of those widely used in macros. It would require many pages to describe how each and every **troff** command is used.

Dimensions in troff

Commands in **troff** recognize dimensions in terms of an inch, a centimeter, a pica (1/6 inch), a point (1/72 inch), an em (the width of the letter *m* in the current point size), an en (half an em), a vertical space (from baseline to baseline of a letter), and a machine unit (which depends on the output printer), represented by the abbreviations *i, c, p, P, m, n, v,* and *u,* respectively. These are summarized in Table 4. Even if this list seems excessive, these units are common in the typesetting business, except for machine units, which were invented for **troff**.

Specifying Page Layout

You specify page dimensions by using **troff** commands for setting the page length, the page offset, the line length, and indentation. The commands for setting dimensions are shown in Table 5.

Obviously, these dimensions are constrained by the physical size of your page. If your printer can only handle 8 1/2-by-11-inch paper, results of commands such as **.ll 9** are unpredictable.

Specifying Fonts and Character Sizes

You choose the typeface by using **troff** commands to specify font and point size (measured in points). Commands for this purpose are summarized in Table 6.

Unit	Abbreviation	Size
inch	i	
centimeter	c	
pica	p	1/6 inch
point	p	1/72 inch
em	m	Width of letter *m* in current point size
en	n	Half the size of an em
machine unit	u	Depends on printer

Table 4 Dimensions in troff

Description	Command	Action	Default
Page length	**.pl** *n*	Set page length to *n*	11.0 inches
Page offset	**.po** *n*	Set page offset to *n*	0.75 inch
Line length	**.ll** *n*	Set line length to *n*	6.5 inches
Indent	**.in** *n*	Indent all subsequent lines by *n*	0.0 inches
Temporary indent	**.ti** *n*	Indent only next line by *n*	

TABLE 5 Commands for Page Layout

Fonts are represented by one or two characters. For instance, *H* refers to *H*elvetica and *CW* refers to *C*onstant *W*idth font. To change to Helvetica, you can use either the **.ft H** command or the inline escape sequence **\fH**. To change to Constant Width font, either use the **.ft CW** command or the inline escape sequence **\f(CW**. You can return to the previous font (the one used before you made the last change) using **.ft P** or **\fP** (*P* refers to the *p*revious font).

Your selection of fonts depends on your output device. In DWB, there is no limit to the number of fonts you can have, although good taste should sharply restrict how many fonts you use within a document. You will need to find out which fonts are supported by your printer.

Using the **.ps** command or the **\s** escape sequence allows you to select a point size. Some output devices, including many laser printers, support all point sizes for each font. However, the range of sizes available may depend on your output device (laser printer, typesetter, or display) and on the font.

Description	Command	Action	Default
Point size	**.ps** *n*	Set point size to *n*	10 points
	.ps *n*	Return to previous point size	
	\s *n*	Set point size to *n*	
	\s + *n*	Increase point size by *n*	
	\s – *n*	Decrease point size by *n*	
Font	**.ft** *f*	Switch to font *f*	Roman
	.ft	Return to previous font	
	\f*X*	Switch to font with one-character name *X*	
	\f*XY*	Switch to font with two-character name *XY*	

TABLE 6 Commands for Changing Fonts and Point Sizes

Placing Text

To control the placement of text, you need a wide range of capabilities. Table 7 summarizes **troff** commands used to start new lines or pages, to turn on and off filling or adjustment, to set spacing between lines, and to center lines.

Escape Sequences for Special Effects

In many situations, you will not want **troff** to interpret your input literally. For instance, when you use an embedded command, such as **\fB**, you do not want **troff** to actually print out the three characters in this string, but rather, you want to change the font to bold. The backslash character is the *escape character* for **troff**; when **troff** finds a backslash, it examines the next characters for special meaning.

A backslash followed by an ampersand (\&) produces a character with zero width—that is, a null character with no space. This can be useful to print out lines that begin with a period. For instance, the **troff** input

```
\&.ce
```

produces the line of text:

```
.ce
```

Description	Command	Action	Default
Line break	**.br**	Start new output line without adjusting current line	
Fill mode	**.fi**	Use fill mode for output	
	.nf	Use no-fill mode and no adjusting	
Adjusting	**.ad**	Adjust margins	
	.na	Turn off margin adjustment	
Centering	**.ce** *n*	Center next *n* input lines	
Break page	**.bp**	Start new page	
Baseline-to-baseline spacing	**.vs** *n*	Set spacing between base lines to *n*	12 points
Spacing between output lines	**.ls** *n*	Add *n* – *1* blank lines to each line of output	No blank lines (*n* = 1)
Space vertically	**.sp** *n*	Space vertically by *n* (downward if *n* > 0; upward if *n* < 0)	
Need space	**.ne** *n*	Skip to next page unless space of *n* is available on current page	

Table 7 troff Commands for Text Placement

You can print out a single backslash by using **\e**. Alternatively, use the **.ec** command to reset the escape character to something you use less often. For instance,

```
.ec  @
```

resets the escape character to @ from the default escape character (\). Alternatively, you can disable the escape mechanism completely using the following command:

```
.eo
```

This makes the backslash act as an ordinary character instead of an escape character.

In all the following examples, the backslash will be the escape character.

Protecting Blank Spaces A blank space preceded by a backslash is *protected*; it will not be padded when the line is justified. For instance, when you include

```
United\ States\ of\ America
```

with protected blanks, the words of this string will not be spread apart during formatting.

Escape Sequences for Non-ASCII Characters Typesetting fonts contain many special characters not among the set of ASCII characters. These include the copyright mark, the cent sign, letters in foreign languages such as Greek, bullets, and other assorted symbols. These characters are inserted using specific four-character escape sequences of the form \(*xy* (the backslash character, followed by the open parenthesis character, followed by the two ASCII characters specified by *x* and *y*). Some examples are shown in Table 8. The particular set of non-ASCII characters that you can print using escape sequences depends on your printer and the software that drives it.

Overstriking

You can build your own special characters using overstriking. The **o** command prints (up to nine) characters on top of each other, with each character centered to the center of the widest character. For instance,

```
\o'Y='
```

produces the symbol for the Japanese yen.

Escape Sequence	Character	Name
\(rg	®	Registered trademark
\(tm	™	Trademark symbol
\(rh	☛	Right hand symbol
\(*a	α	Lowercase alpha
\(*P	Π	Uppercase pi
\(12	½	One-half symbol
\(sq	□	Square

TABLE 8 Samples of Escape Sequences for Special Characters

The zero-motion command, **\z**, can also be used for overstriking. The **\z** command suppresses horizontal motion after a character has been printed, so that all characters are printed beginning at the same place. For instance,

```
\z\s9\(ci\s12\z\(ci\s15\(ci
```

prints three circle characters, of sizes 9, 12, and 15, respectively, beginning at the same place.

Printing Titles

You can produce three-part titles using the **.tl** command. A **.tl** command takes three strings as arguments, separated with apostrophes. The first string is left-adjusted, the second string is centered, and the third string is right-adjusted. For instance,

```
.tl 'Section 1'ANNUAL REPORT'Chapter 1'
```

produces

```
Section 1       ANNUAL REPORT       Chapter 1
```

Conditionals

You can use the **.if** command to have **troff** test whether a condition holds before carrying out a command. The **.if** command has the following form:

```
.if test command
```

For instance, the instruction

```
.if \n%=1 .sp 12
```

produces 12 blank lines if the current page number is 1, and does nothing otherwise. (In this instruction, \ n% is an example of a number register, described later in this chapter.)

There are two important pairs of built-in tests. The first pair, *o* and *e,* tests whether the current page number is odd or even, respectively. The second pair, *n* and *t,* tests whether **nroff** processing or **troff** processing is used. For instance,

```
.if o .tl 'Chapter 1'ANNUAL REPORT'Section 1'
.if e .tl 'Section 1'ANNUAL REPORT'Chapter 1'
```

produces the title

```
Chapter 1       ANNUAL REPORT       Section 1
```

on odd-numbered pages, and

```
Section 1       ANNUAL REPORT       Chapter 1
```

on even-numbered pages.

The **troff** commands

```
.if n .sp 2
.if t .sp 1.5
```

produce 2 vertical spaces when **nroff** is used and 1.5 vertical spaces when **troff** is used.

Defining Strings

You can use a *string variable,* which can be one or two characters, to represent text you use repeatedly. To define the string variable *xx,* use the following command:

```
.ds xx string
```

If there are blank spaces in the string, double quotes are used. After this variable definition is made, **troff** will expand the escape sequence \(**xx* into the value of the string. For instance, since the string "UNIX\ ® System V Release 4" is used often, a variable can be defined to have this string as its value:

```
.de U4 "\s-1UNIX\s+1\(rg System V Release 4"
```

The input

What is new in \(*U4?

is expanded by **troff** into

What is new in UNIX\ ® System V Release 4?

Care must be taken when choosing the name of a string. Do not use the names of **troff** commands, other string variables, or macros you plan to use, since these objects share a common name list.

Number Registers

Number registers store information about the status of ongoing **troff** processing. There are more than 30 predefined registers storing the values of the point size, the page number, the day of the week, and so on. You may also define and use your own number registers.

The names of number registers are either one or two characters. For instance, the predefined number registers *.i*, *.l*, *%*, and *dy* contain the current indent, the current line length, the current page number, and the current day of the month, respectively.

You can obtain the contents of a number register with the escape sequence **\n(***xy* when the name of the number register is two letters, and the escape sequence **\n***x* when the name of the number register is one letter. For instance, **\n(dy** gives the current day of the month and **\n%** gives the current page number.

You may use these registers as part of a **.tl** command to produce a title:

```
.tl '\n%'UNIX SVR4'\n(mo/\n(dy/\n(yr'
```

This produces the following three-part title (assuming today is 12/31/99):

342   UNIX SVR4   12/31/99

You can use the **.nr** command to reset a predefined number register if its name does not begin with a dot. For instance,

```
.nr % 47
```

resets the number register %, which holds the current page number, at 47.

You can name your own number registers any one- or two-character string you wish (other than the names of the predefined registers), because they do not share the same name space as strings, macros, and commands. To set the value of a number register you define, use the **.nr** command. For example,

```
.nr Zz 3
```

sets the value of your number register Zz to 3.

Obvious applications of number registers include automatic numbering of chapters, sections, tables, figures, displayed equations, displayed command lines, and so on.

Writing Macros

You may often have to use certain combinations of **troff** commands when you format your documents. Or you may need to create sets of **troff** commands to implement certain design elements in a document. In these and related situations, you would like to have customized macros available. To meet this need, you can use **troff** to create your own macros.

Defining Macros

The first line in the definition of a macro is

```
.de xx
```

where *xx* is the name of the macro. The **troff** commands that make up the macro follow. The final line of the definition is .. (two periods).

You can name a macro with any string of one or two characters. Obviously, you should avoid giving a macro the same name as a **troff** command or an existing macro. Also, if you plan to use an existing macro package, you need to avoid using names of macros in the package (including those defined internally within the definitions in this package). One suggestion for avoiding the names of existing macros is to use two-character names, where the first character is lowercase and the second is uppercase, for your macros. Generally this avoids the names of the macros in existing packages.

When developing a macro, begin with a simple version that may not do exactly what you want and may not have all the capabilities you need. Then write refined versions of the macro by replacing **troff** commands with more powerful combinations of commands and by adding new commands.

To show how macros are defined, what follows will define variants of a macro named **.nP**, used to start a new paragraph. By using this name, you can avoid duplicating a name used in the **mm** macro package.

First, look at the definition of a relatively simple version of **.nP**. This version produces two vertical spaces, indents the next line (and only that line) four horizontal spaces, and begins the paragraph on the current page when at least one inch remains on that page, and on the next page when less than one inch remains. Here is the definition:

```
.de nP
.sp 2
.ti +4
.ne 1.0I
..
```

We can make **.nP** more general by using a number register. For instance, the definition

```
.de nP
.sp 2
.ti \n\(.i
.ne 1.0I
..
```

does the same thing as before, except the indentation of the first line is *.i,* the value of the number register holding the current indent.

You will also find conditionals useful within macros. For instance, you can refine the **.nP** macro to handle **nroff** and **troff** differently:

```
.de nP
.if n .sp 2
.if t .sp 1.5
.ti \n\(.i
.ne 1.0i
```

This macro produces 2 blank vertical spaces when **nroff** is used and 1.5 vertical spaces when **troff** is used. Otherwise, it does the same thing as before. You can define a variant of this macro that will number your paragraphs using a number register to count the paragraphs. First, initialize the number register P# at 1 with the following command line:

```
.nr P# 1
```

Then, define **.nP** as follows:

```
.de nP
.sp 2
.ti 4
Paragraph \\n(P#
.ne 1.3I
.nr P#+1
..
```

When you use the preceding macro, each of your paragraphs will start with a header, Paragraph *n*, where *n* is the number of the paragraph. The line .nr P#+1 increments the value of the number register P# by one. An extra slash is used before \ n(P# because the number register P# needs to be evaluated on the second pass. The first pass occurs when **troff** reads the macro definition.

Arguments for Macros

You can also give arguments to macros to make them more flexible. You use \\$1, \\$2, \\$3, and so on, to get the values of the first argument, the second argument, the third argument, and so on. You use double slashes before the *$n*'s so that **troff** will read in their values on its second pass. This is needed, because the first pass is used to define the macro; the macro is used, wherever it occurs, on the second pass. For instance,

```
.de nP
.sp \\$1
.ti \\$2
.ne 1.0I
..
```

is a macro that takes two arguments; the first specifies the spacing before a new paragraph, and the second specifies the temporary indent for the first line. The instruction

```
.nP 1.1i 5m
```

produces 1.1 inches of blank vertical space and a temporary indent of 5 em spaces each time it is used.

You should be able to build useful macros using the material introduced in this book. However, you will need to consult references listed at the end of this chapter to find out all the features of **troff** that are available for writing macros. Here are some general warnings before you try:

- It is harder to write macros to do what you want than it looks.
- Someone else has probably done it already.
- When in doubt, try another backslash! (And remember that sometimes you will need to add pairs of backslashes or even worse, sometimes the number of backslashes you will need will be a power of two!)

Using Source Files

You do not actually have to include macro definitions in your text files each time you wish to use them. Instead, you can keep your private macros in a separate file and use the **.so** (*so*urce) command to read them in when processing occurs. In general, the **.so** command reads the contents of another file into the input stream. Once the contents of this file have been read, the current file is once again used as the input stream. For instance, if you have a macro package in the file *newmacros,* you can put the line

```
.so newmacros
```

at the beginning of your file, and the definitions of your macros will be read in when **troff** processing is carried out.

The **.so** command can also be used when putting together files for large projects. For instance, the file *book,* for a publication, may contain the following lines:

```
.so Preface
.so Contents
.so Chapter1
.so Chapter2
.so Chapter3
.so Chapter4
.so Appendix
.so References
.so Index
```

The contents of each part of the book are formatted in separate files, which are given as a series of source files. The entire book can be printed without having to remember the order of the files that make up the book.

A difficulty arises with this approach when preprocessors are used. For instance, the command

```
$ eqn book | troff -mm | lp
```

would not work properly, because **eqn** would only operate on the actual contents of the file *book,* before the **troff** program incorporates the contents of the source files. To get around this problem, you must use a command that pulls source files in. This can be done using the **xpand** command. (On older systems, use either the **soexpand** or **soelim** command instead.) You print the document using the following command line:

```
$  xpand book | eqn | troff -mm | lp
```

Where to Find Macro Packages

For **troff** to use a macro package, the definitions of its macros must be available in the file *tmac.mname* in the directory */usr/lib/tmac,* where *mname* is the name of the macro package. For instance, **troff** looks for the definition of the **mm** macros in the file */usr/lib/tmac/tmac.m.*

On some systems, files in */usr/lib/tmac* do not actually contain macro definitions. Instead, they contain **.so** commands that switch the input stream to the file where these macros are defined. For instance, on some systems, definitions of the **mm** macros are in the file */usr/lib/macros/mmt,* so that the */usr/lib/tmac/tmac.mm* contains the line

```
.so /usr/lib/macros/mmt
```

Macros are always readable ASCII text. Before you try writing a macro, swipe one that almost does the job from a standard macro package or from a colleague and modify it.

Printing Form Letters

Producing form letters also requires the use of another file. For instance, you may need to send customized copies of a letter to hundreds of people. You can use the **.rd**, **.nx**, and **.ex** commands to produce form letters. You use the **.rd** (*r*e*a*d) command to have a **troff** file read input from standard input. You use the **.nx** (*n*e*x*t) command to switch processing to a specified file, and the **.ex** (*ex*it) command to tell **troff** to stop formatting.

Following is an illustration of how to use these commands to send a subscription renewal letter. The file containing text and format commands follows.

```
.PH ""
.DS
.rd

.rd
:
.DE
This letter is to advise you that your subscription to
the UNIX Intergalactic Newsletter is about to expire.  To
renew your subscription, please send $100 to us by the
end of this month.
.sp
.ce
Sincerely yours,
.sp 2
.ce
Oscar Gupta, Editor in Chief
.bp
.nx renewal
```

You put the information to be read in the file *list.* You use a block containing the name and address lines, followed by a blank line, and then the salutation, which is followed by a blank line. Then you put in the next name and address, a blank line, a salutation, and so forth. You end this file with a blank line followed by the **.ex** command. For instance,

```
Dr. Alicia Adams
Software Video Corporation
1001 North Main Street
Hampden, Maine 04444

Dear Dr. Adams:

Robert Ruiz
Room A-123
Department of Computer Science
University of New Jersey
Grovers Corner, New Jersey

Bob,

.ex
```

To print the form letters, use the following command line:

```
$ cat list | troff -mm renewal | lp
```

A separate letter will be printed for each person on the list.

The troff Command and PostScript

The PostScript Page Description Language is used extensively for generating output for laser printers. You can print **troff** output on PostScript printers using the **dpost** command. You can also include pictures formatted using the PostScript Page Description Language with some special macros. These capabilities are described next.

Printing on a PostScript Printer with dpost

You can print documents formatted with **troff** on any printer that supports the PostScript Page Description Language. To print out a document formatted with **troff** on a PostScript printer, you need to convert the output of **troff** to the PostScript Page Description Language. You can do this using the **dpost** program included in Documenter's Workbench 3.0. You'll use this command line:

```
$ troff -mm -Tpost file | dpost | lp
```

(Some versions of **lp** are smart enough to call **dpost** automatically if they get **troff** output.)

Once your **troff** file has been converted to the PostScript Page Description Language, you can modify what your output will look like by making changes in the PostScript file. You have to understand the PostScript language to be able to do this. For instance, you can change textures and shadings, you can insert new images, and you can even define and use your own fonts.

Including PostScript Pages in Your Documents

You can include pages described in PostScript in your documents formatted with **troff**. Of course, you have to have some way to create the PostScript file describing your image. This can be done using a variety of popular software packages on any type of system that can send a file to your UNIX System, such as a Windows PC, a Macintosh, or a UNIX System with graphics software.

The inclusion of PostScript pages is done using three macros (in the *mpictures* macro package). Usually you only need to use the macro **.BP** (*b*egin *p*icture). This macro takes the name of the file containing a page description in PostScript code and arranges for the page to be printed in a space defined by a height and width, with the given offset relative to the given horizontal position at the current vertical position. Also, text will fill any space around the picture. The format of this command is

```
.BP file [height] [width] [position] [offset] [flags] [label]
```

For instance, a PostScript page containing a depiction of the U.S. flag was included using the following command:

```
.BP flag.ps 2 3 r 1 "A PostScript Picture"
```

This **.BP** command printed the page described by the PostScript file *flag.ps* in a two-by-three-inch space, one inch from the right margin, with the label "A PostScript Picture" printed under the picture, within the page of **troff** output. (Note that the default units for **.BP** are inches.) The default height used by **.BP** is 3 inches, and the default width is the length of the current text line (line length minus indent). The position options recognized are **l**, for the left end of the current line of text, **r** for the right end of the current text, and **c** for the center of the current text.

To print the same picture with a box around it, use the **o** flag:

```
.BP picture.ps 2 3 r 1 o "A PostScript Picture"
```

The **.EP** (*e*nd *p*icture) macro is used to end the text filling. The **.PI** (*p*icture *i*nclude) macro is a lower-level macro used by **.BP**. You can use **.PI** together with **troff** commands to do things that .**BP** cannot do.

To print out a file containing PostScript pages, use the following command line:

```
troff -mm -mpictures -Tpost file | dpost | lp
```

CAUTION *Some systems implement* **lp** *via a network-attached print spooler. If you count on* **lp** *to invoke* **dpost** *automatically, it may get invoked on a background processor that does not have access to your PostScript file. For this reason, always invoke* **dpost** *explicitly.*

Previewing troff Output using the X Window System

You can preview **troff** output on your screen if you are running the X Window System on your terminal. For example, you can preview your **troff** output if you are using OPEN LOOK on the X Window System (see Chapter 26) and have the **xhbt** program that is part of DWBX 3.4. You can display the **troff** output of a file containing **grap** code, **pic** code, **tbl** code, **eqn** code, and **mm** macros on your display using this command:

```
$ grap file |  pic |  tbl  |  eqn  |  troff -mm |  xhbt
```

The **xhbt** program is based on the **xditview** program that is distributed with the X Window System Version 11, with numerous enhancements for working with the **troff** system, including several aesthetic improvements. **xhbt** can also be used to turn existing documents formatted using **troff** into hypertext documents. See the volume devoted to DWBX that is part of the Documenter's Workbench Release 3.4 documentation for details.

Another way to preview your documents formatted using the **troff** system is to use **ghostscript**. **ghostscript** is a commonly used tool for displaying PostScript files on an X Window terminal. However, **ghostscript** does not contain the special features of **xhbt** designed to produce optimal output for displaying documents formatted using **troff**. Because of this, **troff** documents displayed using **ghostscript** will not have the sharpness produced when **xhbt** is used, especially when tables and equations are present. If you do not have DWBX 3.4, you might want to use **ghostscript** to preview documents produced with **troff** on your X Window terminal. You can obtain **ghostscript** from GNU archive sites.

Summary

You have seen descriptions of many of the capabilities of the **troff** system for advanced typesetting tasks. Several different **troff** preprocessors for formatting tables, equations, line drawings, and graphs have been introduced. To master these preprocessors, you should work through many examples (consult the appropriate references listed at the end of this chapter and the chapter "Introduction to Text Processing"). Once you have become adept at using them, you will find them relatively easy to use.

You have been introduced to many **troff** commands and constructions. You have seen how to lay out pages, how to do many special types of formatting, and how to write macros. You will find that mastering **troff** sufficiently well to customize documents is much like learning a programming language. However, it is well worth your effort if you need to format a wide range of documents with diverse layouts and styles. You will need to consult the appropriate references for a complete list of **troff** commands.

Finally, you have seen how to print documents formatted with **troff** on PostScript printers. You have learned how to include PostScript pages in your documents formatted with **troff**. You have also learned how to preview documents produced using the **troff** system on your terminal if you use the X Window System.

How to Find Out More

To find out more about the topics covered in this chapter, you can consult the reference given at the end of the chapter "Introduction to Text Processing" as well as the following documents. Note that many of them are out of print but may be available through used book stores or through collectibles services such as eBay.

Bentley, J.L., and B.W. Kernighan. "grap—A Language for Typesetting Graphs: Tutorial and User Manual." *Computing Science Technical Report 114*. AT&T Bell Laboratories, 1984.

Kernighan, B.W. "A troff Tutorial." *UNIX Programmer's Manual* (7th ed.), Volume 2. AT&T Bell Laboratories, Holt, Rinehart, and Winston, 1983.

Kernighan, B.W. "pic—A Graphics Language for Typesetting: Revised User Manual." *Computing Science Technical Report 116.* AT&T Bell Laboratories, 1984.

Lesk, M.E. "tbl-A Program to Format Tables." *Computing Science Technical Report 49.* AT&T Bell Laboratories, 1976.

Ossanna, Jr., J.F. "nroff/troff User's Manual." *Computing Science Technical Report 54.* AT&T Bell Laboratories, 1977.