Adventures with Typesetter-Independent TROFF

Mark Kahrs† and Lee Moore
Department of Computer Science
University of Rochester
Rochester, NY 14627

TR 159
June, 1985

Abstract

This report describes our experiences with Typesetter-Independent TROFF. War stories are related regarding the writing of an output filter for an experimental laser printer (Xerox Press) and for a strange phototypesetter (Compugraphics 7700). The suitablity of TROFF to these tasks is discussed as well as bugs and traps that were found on the way. Hints are given for future implementors of TROFF post-processors. Portions of this paper previously appeared in the Proceedings of the 1984 Summer Usenix Conference.

---

†The work was performed at the University of Rochester. The first author's current address is: AT&T Bell Laboratories, 2C-455, Murray Hill, NJ 07974

1. Introduction

At the University of Rochester we have access to three "oddball" output devices – two experimental Xerox prototype laser printers and a Compugraphics Editwriter 7700 model II phototypesetter. Therefore, when the availability of Typesetter-Independent Troff (sometimes called "device independent TROFF") was announced, we were naturally excited. This report relates what we have learned about (1) the new version of TROFF and (2) our printers. We offer some advice to future writers of TI-TROFF output filters based on our experience.

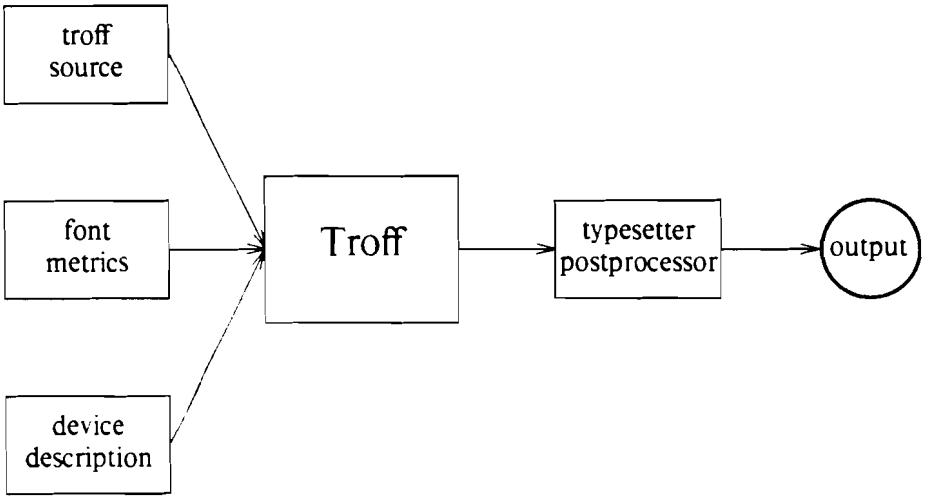
We will first review the TI-TROFF organization and then we will discuss the implementation for the PRESS output converter for the Xerox laser printers. Later we will discuss the implementation of the output filter for the Compugraphics Editwriter 7700 and then conclude with some general remarks on TI-TROFF.

This report was created using the software described below.

2. TROFF

As most readers know, TROFF [ker78] is a descendent of an original runoff done for CTSS [cri65]. The typesetter-independent version was created by Brian Kernighan from the original version of Ossanna and subsequently reported in a Bell Labs Report [ker81]. Kernighan's report makes interesting reading as he details the decisions that led to the design of the common output language as well as the font information files.

The new TROFF has the following organization:



The new TROFF takes an input file and information about the target device and produces a text file of commands. The format of these commands is common to all devices and must be transduced to device specific form by a separate program. When re-targeting TI-TROFF for a new device, one must create the device description file and write the post-processor.

2.1. Input to TI-TROFF

When TI-TROFF starts execution, it loads a file that contains critical information about the output device. Basically, this file contains the resolution of the output device in "goobies" per inch, the vertical and horizontal positioning resolution, the "unitwidth" (the font size at which all the character widths are given), the paper width and paper height (for those output devices without rolls) and finally, the list of font names.

Having TROFF do its computing in the units of the target machine eliminates the roundoff errors that occur during the scaling of coordinates. In the past, people have tried to simulate the actions of the C/A/T phototypesetter by translating C/A/T opcodes into actions on other devices. The 200 spot per inch (s.p.i.)

Versatec printer/plotter has historically been the most popular target. Since none of these output devices had the same resolution as the C/A/T (432 units per inch) round-off errors were common.

In TI-TROFF the horizontal and vertical resolution are specified in goobies. That is to say, TI-TROFF lays a fixed grid on the page and expects motions to occur only between the grid points. As we will see last, this seemingly innocent design decision will lead to problems in the case of the Editwriter.

2.2. Output from TI-TROFF

The output of TI-TROFF is a single file of commands of the form <command-letter> <data>. This syntax is disobeyed in one special occasion to achieve data compression (However, this saving goes away when the output device has a high resolution!). The formats are as follows:

s*n*	size in *n* points
f*n*	mount font *n*
c*x*	display character *x*
C*xy*	display character \(*xy*
H*n*	go to absolute horizontal position *n*
V*n*	go to absolute vertical position *n*
h*n*	go to relative horizontal position *n*
v*n*	go to relative vertical position *n*
nnc	[cheat] move horizontally *nn*, then display *c*
n*b a*	new line, *b* is the space before the line, *a* is after
w	word space (after a word)
p*n*	start page *n*
x ...\n	special device functions (x is for excape)
D ...\n	drawing functions

The output from TROFF is "complete", i.e., it already has horizontal moves and spacing information included in it. This will be shown to be a problem with "smart" typesetters such as the Compugraphic 7700 or PRESS style devices.

2.3. TROFF Design Quirks

In this section, we will describe design quirks of TROFF that we felt reduced its typesetter independence. Most of the warts exist because TROFF was originally targeted for an inexpensive phototypesetter, the Wang (née Graphics Systems) C/A/T. Although TROFF has shown remarkable adaptability it still has many birth-marks.

For example, it assumes that the device has all the sizes of a given font. In an analog phototypesetter where point size changes are made by manipulating a lens, this is a quite reasonable assumption. However, in a digital phototypesetter it is common *not* to have all the fonts in all the point sizes.

Another C/A/T phototypesetter assumption is that the carriage doesn't move after a character is flashed. Therefore, a horizontal move must be generated to move over the last character. For both of our output devices this turned out to be inappropriate. This will be discussed further in section 3.5.

Let's discuss what special characters mean in TI-TROFF. The C/A/T phototypesetter was not designed with ASCII in mind. The non-special fonts (R, I, B) had non-ASCII characters as well as *not* having many normal ASCII characters. The folks at AT&T Bell Laboratories compensated for the missing characters by putting them on the special font. Because they were special, characters such as @, #, <, >, and " were invariant to font changes. That is to say, they looked the same regardless of whether you were in roman, bold or italic. We had a fear that if we didn't simulate this design quirk, then old documents wouldn't come out correctly. In the end, we choose to do the "right thing" rather than emulate the ways of old ghosts. This worked out in the end, but we did discover that math plus, math minus and math equals were different from the ascii plus, minus and equals. The math versions are assumed by EQN to be always in roman and not in the current font. In addition, the eqn manual gives many users the impression that by typing the string:

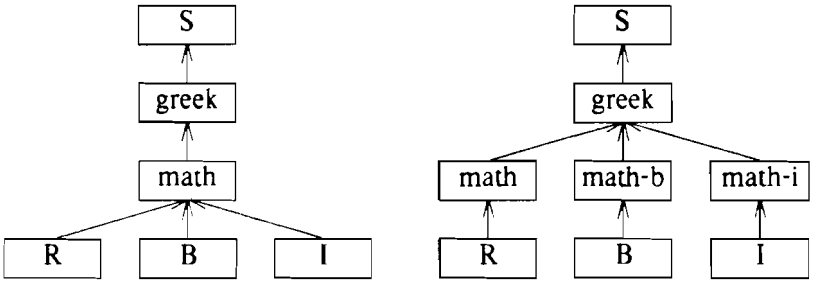
"{"

one can get a roman "{". In fact, the quoting actually means asking eqn not to process that string at all. Since eqn sets equations in italics by default, the "{" will come out in italics. The manual shows it in roman because the C/A/T can only print the "{" in roman. This is because that character only exists on the special font.

Since the C/A/T phototypesetter only had one font family (Times Roman), the concept of font family never developed. It is not possible to ask for the bold version of the current font because TROFF doesn't really "understand" that a given font is bold or italic or whatever. Nor is it possible to associate sets of special characters with particular fonts. For example, our laser printer had bold math symbols which we would have liked to associate with the bold fonts. Although we had to forget about automatic usage of the bold math font, the use of font families is partly eased because of the convention of using font positions instead of font names. This feature is found in both the "ms" and "me" macro packages and works because these packages assume that the roman, italic and bold versions of a font are mounted on positions 1, 2, and 3 respectively. Thus to change families, mount the new family on these positions correctly and most macros still work. One difficultly that remains is that normal TROFF macros don't understand the idea of bold-italic fonts.

While we are talking about special characters, it should be noted that TI-TROFF has a limit of 221 special character names. If you have more, TI-TROFF will give you it's usual diagnostic: a core dump.

Let's get back to our problem with the bold math font. TI-TROFF relaxed the concept of the special font so that it was no longer just a single physical font on the output device but a collection of physical fonts. This collection is searched when the current font doesn't have the desired character. So while the special font became multiple physical fonts, each regular font was still mapped one-to-one to a physical font. It is easy to find cases where is it is desirable to map several physical fonts to one logical font. One motivation comes from the fact that many typesetters don't have full ASCII in any one font. Below we diagram the inheritance network for the current TROFF and the network we would like to create if we had a more flexible system.



It should be noted that others have hacked around this problem by doing look-ups in the post-processor. In any case, we decided to forego using the bold math font because of the poor cost-benefit ratio.

Our laser printer also came with the ability to print out text at multiples of 90°. Unfortuately, TROFF doesn't offer a means of specifying rotated text. Naturally, the C/A/T didn't rotate text either. We made a version of our laser printer filter that would print a whole document in landscape mode (11×8½ as opposed to 8½×11). In addition to changing the filter, we also exchanged the values of `paperwidth` and `paperlength` in the DESC file. While testing we discovered that TROFF ignored the `paperwidth` and `paperlength` fields of the DESC file. It was therefore impossible to set the line length beyond approximately 7.54 inches. Thus we were unable to create extra wide tables which was the motivation for this feature.

Not all devices have a fixed coordinate system like the one in TROFF. The Editwriter has a coordinate system that varies as a function of the current point size. This is quite handy for simple things because it means that you can request the typesetter to go to the next line and it will be positioned correctly because the height of a line relates to the current point-size.

The ultimate destination of a TROFF document can be any number of output devices and each device can have wildly different font sets. In our case, the two Xerox laser printers have very different font sets at very different resolutions. Unfortunately, TROFF doesn't allow you to describe a device class and then talk about instances of that class.

Although the description file for the typesetter was supposed to be the "source of all knowledge" about the output device, the filter packages (TBL, EQN, PIC) don't look there. All of the device dependent information is hard coded into the C source of all these filters.

3. War Stories about TROFF

Herein we relate actual events as they happened to your intrepid reporters. These stories relate to our struggles with the beast TROFF itself and should be of interest regardless of the choice of output device.

When we looked at the some of the early output from the X2600, a few of the characters were overlapping. Furthermore, the same characters would always overlap. Then we noticed that it would only happened to the widest characters: like "M" and "W". Why is it, we asked ourselves, that some of the widest characters are treated like they have the thinnest widths? Well, it turns out that the maximum width of a character in a compiled font description file is 255 goobies at unitwidth, which was not documented *anywhere*. Should a character in the input description file be greater than 255, its high order bits are (silently) truncated. For both of our devices we had to specify widths using rather large cardinal numbers. Fortunately, all the widths were less than 512 so we just halved the unitwidth and re-computed the input tables. (The TROFF tables were automatically generated).

Another one of our early test documents caused TI-TROFF to bomb out with the enlightening diagnostic: "floating exception". Naturally we were quite puzzled by this since the document worked fine with old TROFF. A good nights sleep and a short session with sdb showed that the exception took place when a value was being divided by the width of the ruling character. Since we hadn't defined a ruling character yet, the width was zero and hence the exception. Needless to say, we quickly added a baseline rule.

When we tried out our first font table on TROFF we got error messages of the form "Can't find font: 1.out". This is, of course, rather strange because the font in position 1 should have been converted to its name before being looked up in the font directory. It was later discovered that TROFF depends on the existence of at least one special font (any one, doesn't much matter which one) when converting from font numbers to font names. Since our initial device description didn't include any special fonts, we tickled this bug. The actual location of this bug is in the routine findft(). In TI-TROFF there is a variable called *smnt* which is the number of the first special font. If there isn't a special font, then this variable is set to zero (0). Below is the expression that checks to see if the variable *i* is a font position. Note that the expression is always false if *smnt* is zero.

```
if ((k = i - '0') >= 0 && k <= nfonts && k < smnt)
```

If you don't plan on having special fonts, you might want to change this expression to something like:

```
if ((k = i - '0') >= 0 && k <= nfonts && (smnt == 0 || k < smnt)
```

In our first implementation of the PRESS converter, we used the "w" command (white space) to indicate that the word (character) buffer could be spilled. This worked quite well until we tried using hanging paragraphs. It turns out that TROFF considers tabs to be characters and *not* white space! Therefore, the end of the word was never caught and the justification for the line was disturbed. This problem was surmounted (see section 3.5), but only at the cost of some ugliness.

Another subtle confusion is that the "−" character on input is really the \(hy character on output while the \− character is the real "−". If one is to use auto-hyphenation, one must define a hyphen character. Without a hyphen character, words are still broken but the place where this occurs is not marked.

When trying out EQN for the first time, we noticed that not only was the spacing off in many places (which we expected) but that certain characters had totally disappeared (which we didn't expect). This problem totally stumped us. We are grateful to Brian Kernighan who finally solved the problem by pointing out that we hadn't defined the \| (1/6em narrow space) and \^ (1/12em half-narrow space) characters.

We felt it was very strange to define these spacing characters in the font definition when they weren't in the fonts but we did and it worked. It turns out that when you give them device code zero (0) they will be swallowed by the standard post-processor.

When trying to use the PIC program for the first time, we discovered that if we tried to do any non-trivial pictures, the preprocessor would complain that the picture was too big. This was traced to a bad test in "pltroff.c". In the old TROFF motions were encoded with a maximum distance of 8192 (2^{13}) goobies. This was later loosened in TI-TROFF to 32768 (2^{15}) goobies. Because our version of pltroff.c hadn't been updated, we received the error message. Below are the old and new lines:

```
/* internal troff limit: 13 bits for motion */
if (xconv(xmax) >= 8192 || yconv(ymax) >= 8192) {

/* internal troff limit: 15 bits for motion */
if (xconv(xmax) >= 32768 || yconv(ymax) >= 32768) {
```

Our 4.2bsd system also complained that there wasn't a closing #endif in this file. Apparently older C compilers don't mind if a file ends without all the conditional compilation statements closed. Putting an extra `#endif` at then end of the file solved this problem. At this point, it should also be mentioned that pltroff can act as a plot to to troff converter with just a small amount of hacking. This allows one to include plotted output in one's documents.

As is often the case with C programs, there are certain arrays in TROFF whose size is fixed at compile-time. This story revolves around a `#define`'d constant called NFONT. The value of NFONT is the maximum number of fonts that can be used. This constant is separately defined for makedev and TROFF. If one tries to use makedev with a DESC file that describes more than NFONT fonts, then the results will be un-predictable. If one tries to run a TROFF whose NFONT is lower than the NFONT in the makedev used to create the fonts descriptions, then the results will be unpredictable. Un-predictable results are segmentation faults, bad output or error messages from left-field.

A subtle bias in TROFF comes from the fact that the original implementers had only model of each typesetter. The problem comes to the fore when different models of the same typesetter are configured differently. In our case, the hardware was the same and the machine input language was the same across both machines but the fonts were somewhat different. TI-TROFF doesn't give you a clean mechanism to separate details about the input language from the device configuration. Thus one is forced to treat machines as configuration-language pairs rather than a language which can be input to many different instantiations. While this problem can arise with a traditional phototypesetter, it became acute with our Xerox printers. At one level we wanted to treat all Xerox PRESS devices the same because they all accept the same input language; on the other hand, different printers have different fonts sets and must be treated differently at certain points. For example, EQN has a hard-coded switch on device type which sets the minimum point size of the device. Since each PRESS device can have a different font set and thus a different minimum point size, the correct solution would have been to declare each different printer as a different device. Partly to retain our sanity, we chose not to do this but it is clear what the problem is.

Last on our list is a problem with the 4.2bsd Unix[1] macros. We discovered that the standard macro packages adds cut-marks to the top of every page. On roll paper output devices, cut-marks are used to indicate page boundaries for cutting. On a device that produces cut pages, these cut-marks are blemishes. To remove cut-marks in -ms, add the following lines to the from of your file:

```
.de CM
..
```

and in -me

```
.de @m
..
```

[1] Unix is a trademark of AT&T Bell Laboratories

4. Implementation 1: Press

This section will discuss the actual printers (the X2600 and Warlord) and how the TROFF converter was written. We will also discuss a few problems that we met and conquered along the way. We will first describe the particulars of our output devices.

4.1. The X2600

The X2600 is a one of a kind prototype that was donated to the University of Rochester, Computer Science Department by the Xerox Webster Research Center. The print engine is a converted 2600 copier (a small desk-top model) that was modified to include a galvanometer and a small Helium-Neon laser. The resolution of the device is 240 spots/inch both vertically and horizontally. A little multiplication shows that there are (8.5)(11)(240)(240) = 5,385,600 points per page.

The printer is driven by a Xerox Alto [tha81] (not sold in stores) that sends control and video signals to the print engine. The Alto came with a set of programs to receive PRESS formated files from the Ethernet[2] and print them. To print a file, the Alto must convert each page to a bitmap, start the 2600 rolling and then shove the bitmap out the door *in real time*. Unfortunately, because of memory constraints in the Alto, this bitmap must be constructed on the (slow) Alto disk. Each page of text typically takes 30 seconds to format and print, although graphics can cause it to take *much* longer.[3]

4.2. The Warlord

The Warlord is another prototype donated to the University CSD by Xerox. The printing engine is an XP-12, the same device found in a model 2700 printer, the DEC LN01 and the QMS Lasergrafix 1200 (tm). The XP-12 is connected to a custom interface called a "DoLI" that converts the output of the Alto OrBit cards (see [tha81] for a description of the OrBit) to the video format of the XP-12. Note that the OrBit is totally different than the interface used by the 2600; in particular, the formatting is done in the OrBit memory *on the fly* which means that certain pictures (like music) can be too complex for the printer to print. On the other hand, it is faster for most applications. Note also that the resolution of the XP-12 was be jacked up from 300 s.p.i. to 384 s.p.i. by hacking the copier electronics.

The next section describes enough of the PRESS format to give the flavor of the conversion process.

4.3. PRESS

The PRESS format[4] was designed to be a "lingua franca" of printers inside of Xerox. While not used in products, it has been remarkably versatile and it has been used by many processors, programs and printers.

PRESS files basically describe a series of pages to be printed. Each page may have text and/or graphics that can be positioned anywhere on the page. Graphics may be either rectangles (which are supported by virtually every PRESS printer) or they may be defined by outlines (which are not supported on many printers). To obtain maximum generality, we tried to use only text and rectangles as much as possible.

Coordinates in PRESS files are given in *micas* (a mica is 10^{-5} meters (10 microns or $\frac{1}{2540}$ of an inch). The alert reader will realize that the input language has a resolution which is about an order of magnitude greater than the true resolution of either of our PRESS printers. This implies that the true resolution of the output device is irrelevant in the PRESS world. One actually pretends that one is printing on an idealized output device.

The origin lies in the lower left hand corner of the page with X values increasing to the right and Y values increasing in the upward direction. This is partially in opposition to TROFF where Y units increase in the downward direction.

---

[2] Ethernet is a trademark of Xerox Corporation

[3] Rumor has it that a certain piece of sheet music took 20 minutes to format!

[4] The (gentle) reader should not confuse PRESS with Interpress, the announced Xerox printing standard.

PRESS has a notion of the current position on the page. The position may be set at any coordinate position. The X and Y coordinates are set separately. When a character is displayed, the current X (and Y) position is always updated. As mentioned before this does *not* match TROFF's typesetting model. PRESS also has commands that image either an individual character or a string of characters at the current position in the current page. This feature will become important later when we discuss the translation of TROFF intermediate "code" into PRESS.

4.4. Constructing the Width Tables

Like most typesetters, the actual representation of the fonts is stored at the printing device (in this case, the disk of the Alto). Application programs need only to know the name of the fonts on the printer, the width of each character in each font and how to construct a PRESS file. This is done by means of a binary file called "fonts.widths". It contains a dictionary of fonts with relative pointers to the specific width tables. The widths are of two types: those specific to a particular family/face/rotation/size and those that are scaled to represent all sizes in a particular family/face/rotation. It turned out that all the popular fonts were stored in scaled format. When scaled, the size is stored in units of thousandths of a point. It is converted to micas with the following formula:

$$Width_{micas} = \frac{1}{1000} \frac{desiredPointSize}{1} \frac{scaledWidth}{1} \frac{2540 micas}{inch} \frac{inch}{72 points}$$

Clearly, it was desirable to create the font description files for TROFF automatically rather than build them by hand. The following catalogs some of our experience. Initially we had some problems because the fonts.widths file had many more fonts in it than our printer actually had. In addition, most of the fonts that we did have were declared to have more characters than really existed. We can only assume that our fonts.widths file was actually one that belonged to another printer and that it only worked for us because it described printer fonts that were derived from the same spline representation that ours were. As a side note, most of the major fonts are compatible across all the experimental laser printers despite differences in actual resolution. PRESS files made at Xerox as well as those files from other Xerox grant universities tend to print as they should.

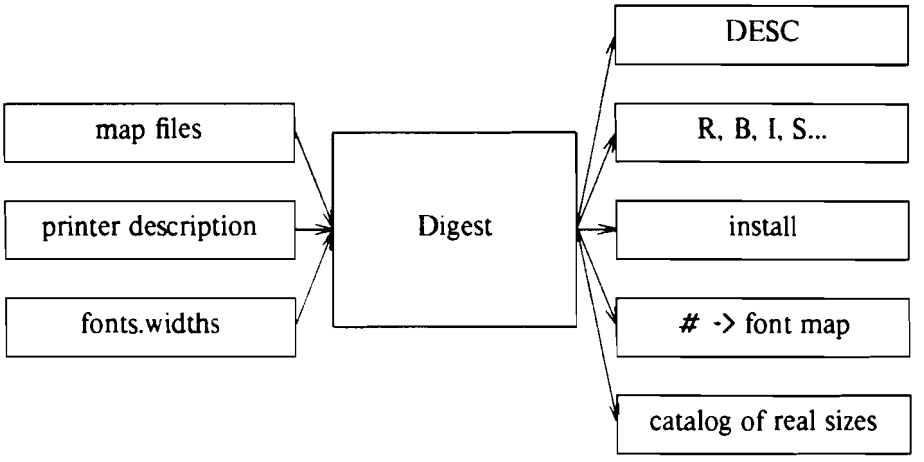
Since most of the fonts in fonts.widths were stored in scaled format, we couldn't automatically determine which point sizes were really available on our printer. To generate a correct device description file (DESC) file, we had to augment this file with additional information.

As mentioned before, all the fonts aren't available in all sizes (not even within a family.) Unfortunately, TROFF has a lens-and-mask model of a typesetter (as discussed previously in 2.3). In this model, the point sizes are determined by the position of the lens and thus the size is font independent. In TI-TROFF, the list of sizes is given in the description file (DESC) and thus is assumed to be uniform across all fonts. We decided to take the union of all font sizes and put that in the description file and to pass the true sizes to the output filter. The output filter compares the point size asked for with the available point sizes and chooses the greatest size less than or equal to the requested size.

Special characters also represented a problem because they had to be mapped from a position in a font to a TROFF name. Fortunately, there were only two fonts with special characters: hippo (greek) and math. To treat this problem uniformly, description files were created that specified the mapping of characters in those fonts to the TROFF character names. Since all fonts in a particular family had the same mapping, the following algorithm was used: first a directory lookup is done on the filename (concat FamilyName ".map"). If that isn't found then the file "default.map" is used. The file "default.map" is basically the ASCII character set. Using the map files also allows us to exclude those characters that are mentioned in the fonts.widths file but don't exist on our printer. The ascender and descender information is also found in the .map files.

Lastly, the conversion program (called "digest") always creates a file describing the pseudo-font "S". This font doesn't really exist on the printer but contains those symbols that that the post-processor will create using the PRESS drawing commands. The bracket building and ruling characters fall into this category. (See Appendix B for the description of these characters).

The "digest" program accepts the fonts.widths file and the .map files for the appropriate device as input. It produces all the font files and the DESC file. In addition, it produces two text files that the output filter (called "dpress") uses to decide what fonts and what sizes are actually on the output device. One important side effect of "digest" is that it preserves the font order of the device description input file in the DESC output file. This is needed because one wants to insure that the Times Roman font (or a reasonable facsimile) is mounted on positions 1, 2 and 3. Naturally, this is for compatibility reasons with macro packages that address fonts by number instead of by name.



Because the DESC file does not describe the output device in enough detail for our application, the "digest" program creates two auxiliary description files. Since the internal name of a font must be represented as a (short) character array, it was not possible to represent the complete family name/face/stress/rotation for an arbitrary font. Also comparisons for equality are expensive in this full n-tuple representation. Instead we chose to internally name fonts by small integers and then lookup the full name at output time. In addition, by having the number of fonts stored in a file, it is an easy matter to allocate the various arrays at run-time rather than using a compiled in "maximum" constant. Another problem was that the output filter needed to know which fonts really were on the printer and which fonts were not. This information is kept in a separate file so that the PRESS subroutine package could be cleanly separated from any influence of TROFF.

The map description files have the following syntax:

{normal | special}
{{octal value} {kern} {TROFF name}*\n}*

An example file is below:

```
#
#  Default mapping of characters
#
normal
#
#       1 = ascender
#       2 = down
#       3 = both
#
# shouldn't define a space (040)
#
#char (a|de)scender      alias
041    1       !
042    1       "
043    1       #
```

```
044   1
045   1     %
046   1     &
047   1     '
050   1     (
051   1     )
052   1     *
053   1     +
054   2     ,
# almost required to have a "hy"
055   1     - hy
056   0     .
057   1     /
060   1     0
061   1     1
062   1     2
063   1     3
```

The printer description file has the following syntax:

device {device name}
{{family} {face} {TROFF name} {point Size}*\n}*

An example file is below:

```
device x2600

# Timesroman always goes first

timesroman  mr R  6 7 8 10 12 18
timesroman  mi I  6 7 8 10 12
timesroman  br B  6 7 8 10 12 18
timesroman  bi X  6 7 8 10 12

elite       mr ER 10

gacha       mr GR 8 10

helvetica   mr HR 6 7 8 10 12 18
helvetica   mi HI 6 7 8 10 12
helvetica   br HB 6 7 8 10 12 18
helvetica   bi HX 6 7 8 10 12

# start of "special" fonts

hippo       mr HP 8 10
math        mr MA 6 8 10
```

One problem that we weren't able to handle elegantly concerned the square root character. All normal characters in our fonts have extra space around them so that if two are printed side by size, then they look normal. Unfortunately, our square root character was defined this way. This means that if one images \(sr followed by \(rn then there will be a gap between these characters. We had to fake out TROFF by editing the font description file before applying makedev so that the square root character was narrower than it was.

4.5. Translating the TI-TROFF Output

The output from TI-TROFF is given directly to the "dpress" output filter. Fortunately, we had access to a set of routines (originally coded in SAIL by Ivor Durham and converted to C by Tom Rodeheffer) that created valid PRESS primitives. We just interfaced this library with the program and we almost had it right. Almost.

First, we were faced with the problem of TROFF to PRESS coordinate systems. Eventually we settled on TROFF coordinate system so that the coordinate system would remain compatible with the graphic (line, circle and ellipse) drawing routines supplied with the TI-TROFF release. Conversion to PRESS coordinates occurs right before calls to the PRESS package.

Second, we wanted to generate compact PRESS files. In general, TROFF will send a positioning command before every character. The obvious implementation of this is to issue a set-position command for each character so that it will be placed exactly. This was the approach taken by "dcat", an existing program that converted the C/A/T output of the old TROFF. Sadly, this created files that were four or more times longer than then the input text. They were bulky to store and bulky to ship around. In addition, they were difficult to print since they often exceeded the spooling space of the disk on the printer Alto. The solution is easy to see because TROFF doesn't do letter spacing. That is to say, it won't insert space between letters, only between words. We could achieve great compression by using the "show character string" command for each word but this meant that we had to figure out where the word breaks were.

Our first attack at this used the "w" command to delimit words. A character buffer is filled by the characters using the "c" commands. The "w" command empties the character buffer and the following "h" command was used to generate the appropriate "set x" command.

Then we discovered an interaction with tabs. Tabs are characters, not white space. Therefore, TROFF doesn't generate a "w" command after a tab. Without this, it's impossible to get TBL right, much less hanging paragraphs. How did we get around it? The answer is quite simple. Recall that each character generates a "c" command followed by a "h" command. The "h" command moves over the width of the character plus any additional move (such as space between words). Now, if the horizontal move is the same width as the character previously printed, then this width must be redundant and therefore no move is generated. However, if the width isn't equal to the last width, then a "set x" command in PRESS is generated.

The "special characters" of TROFF presented their own problems. Many of them were available on existing fonts but many, notably the bracket building functions, were not. Xerox did provide us with the tools to create new fonts. Unfortunately, on the X2600, it is rather laborious to test new fonts because of the way fonts are stored.[5] Also, locally generated PRESS files that asked for these locally created fonts would not print well at sites other than our own. Our solution was to use C code to emit drawing commands for those characters that could be drawn and to ignore characters like the Bell System logo and the pointing hands. Unfortunately, the design of these characters is *not* specified in the TROFF document. Likewise, the design of the "baseline rule" and the "underline rule" is left undefined. So that other TI-TROFF interface writers will not lose their sanity, we are including the definition of the bracket building characters in Appendix B. Note that most of the drawing code was stolen (shamelessly) from the C/A/T filter written at CMU.

Hand coding was also present in EQN, TBL and PIC. Both programs required changes to their internals in order to run. At a minimum, the resolution of the output device has to be "wired in" to both programs. They all ignore the already existing description file! In the case of EQN, the problem is exacerbated by the fact that EQN *expects* the typesetter to have a large selection of point sizes. If one's output device is somewhat limited, things tend to look crowded because TROFF will substitute the next larger point size for sizes it doesn't have. One strategy we tried for dealing with this problem was to define all possible point sizes and then let the output filter substitute the appropriate size. For equations, it made the output look spacious rather than crowded. Unfortunately, this had bad side effects if you selected the wrong size in normal text and so it was later dropped.

---

[5] Our new press printer (a Warlord) does allow a client to send private fonts along with a PRESS file. This feature greatly facilitates debugging and tuning fonts.

5. Implementation 2: Editwriter 7700

The Compugraphic Editwriter 7700 (model II) is a rather conventional phototypesetter that is aimed at low volume print shops. Its light path begins with a drum with four "slots" for mountable font film strips. The next stage is a lens system after the font strips that can magnify a font up to 36 points. The Editwriter has a RS-232 interface called the Intelligent Communication Interface or "ICI". The ICI interprets character strings that come from the modem into pseudo keystrokes of the input typewriter. This interpreter is actually just a finite state machine, but it permits the user of the ICI to "pretend" that a typesetter is "at their fingertips" by designing a simple language for input. The important thing to note here is that we forced to make TROFF generate output for what is a human, not a computer, interface.

5.1. The Phototypesetter

The phototypesetter has a number of characteristics that affect the output converter. To begin with, the typesetter assumes that flashing a character advances the x position (like the PRESS device). Second, the positioning of words on the page is not done by explicit control of the coordinates (i.e., there are not commands like "set x" and "set y"). Rather, the phototypesetter assumes that the commands are of a very high level, like "center this". The motion commands are actually spacing commands; they insert space between characters. A space can either be an em, en or "thin" (3/54 of an em) space. This limited notion of spacing conflicts *severely* with the way TI-TROFF looks at a typesetter. While such an interface is clearly designed with humans in mind, it makes the output filter a kludge.

5.2. The Editwriter Interface

First off, we wanted to create a reasonably compact output language for the ICI. Secondly, we wanted to be able to read it. While it wasn't possible to achieve the first goal to the degree that we would have liked, the second was readily obtainable. The input language to the ICI is controlled by a finite-state automata that is sent before the actual text. It controls the mapping from multi-code input sequences to internal command codes. We designed a language with an escape character ("@") that is followed by one, two, or three characters (depending on frequency of use) that specify what character code the sequence represented. The horizontal spacing commands are kept to one letter for the most part for the obvious space saving reason. We have had no problems with this part of the output filter.

5.3. Constructing the Width Tables

After the payment of a small fee, Compugraphics will supply you with a width table for a given font. Unfortunately, they fail to tell you what units the widths are specified in. Normally, one might think this would be an easy question to answer. Not so. It took at *least* two hours on the telephone to track down the answer. It turns out that the font widths are given in "units"[6]. These "units" turn out to be the number of 16ths of an inch when the glyph is photographed at 250 points. For example, a character with 48 "units" of width is 3 inches wide (at 250 points). Naturally, these widths must be scaled by the current point size.

5.4. Translating the TI-TROFF Output

The output of TROFF is fed directly to the 7700 output filter. The hardest problem is the generation of the spacing commands. The 7700 doesn't have any absolute motions; they are all relative. Furthermore, the distance that a horizontal move command travels is a function of the point size. For that reason, there isn't an absolute coordinate system which means *another* mismatch with TROFF. In the DESC file we defined the resolution to be a close approximation to the coordinate system at a font size of one point. Motions are fixed up later in the output filter. As in the PRESS filter, the 7700 filter compresses its output by recognizing words, however it handles motions slightly differently. The translation strategy for this device becomes as follows: The distance of any horizontal move is compared with the width of the last character. If the width is different, then the output filter generates some number of spacing commands (em space, en spaces and so forth) until no more spaces can be fit. The left over "slop" is carried onto the next

horizontal move. The "slop" is reset at the beginning of each line. What matters, of course, is the leftover slop at the *end* of a line. Fortunately, statistics taken show that the typical average slop is around 25/4000's (0.006) of an inch. In fact, this document had a average slop of 0.003 (12/4000) of an inch.

Like horizontal positioning, vertical positioning must be approximated. In particular, the vertical positioning commands deal in points. Unfortunately, TROFF insists in dealing with goobies (whatevers per inch). And it just so happens that the resolution we use for the printer does not divide nicely into whatevers per inch. Therefore, the description file for the typesetter uses an approximation for the vertical move. Naturally, this generates a bit of slop, but we chose to ignore it on the theory that the eye of the reader is less sensitive to vertical positioning. Vertical positioning on the Editwriter is generally done after every carriage return. The amount of spacing is set by a "line spacing" command. The strategy for the output filter is very simple: If the spacing being asked for is the same as the last spacing, then just emit a carriage return. If the spacing is different, then emit the command to change the spacing and *then* emit the carriage return.

In a simple typesetter like the Editwriter, certain TROFF commands must be ignored. In particular, the graphic commands and rules are useless on the Editwriter.

Another problem with the Editwriter is not immediately obvious. Although it's possible to fool the typesetter into getting the spacing correct, it also takes a fair amount of space on the floppy disk to do so! In fact, the sectors on the disk rapidly fill up.

The Editwriter 7700 has more than its share of problems. Besides the problems of interfacing to a high level typesetter, figuring out the width tables and designing the interface language, the machine isn't even connected directly to our CPU. Instead, it's necessary to go through an intermediate, thereby risking various problems along the way. Fortunately, this has not been a major problem, just a pain in the gluteus maximus.

The Editwriter also forced us to discover another problem with TROFF. The symptom of this failure is the message "font # too big for position #", where the #s are numbers. The solution to this problem is to make the binary font width files the same size. This can either be done by hacking the makedev program, by hacking the fonts, or increasing the maximum size of the TROFF internal buffer. We chose to do the latter alternative.

As mentioned earlier, the Editwriter has a hard sectored floppy disk where it stores the incoming data. Each sector is considered to be a file by the neanderthal file system. During computer input the Editwriter will start a new file when the current sector (file) is full. Unfortunately, during output (actual typesetting), all state is lost between files. This means that state information must be reset at the front of each file. The output filter attempts to compensate for this by directing the Editwriter to begin a new file near sector boundaries and then emitting the state information.

6. Conclusion

6.1. Our Subjective Feelings

The first question is ask is: was it worth it? Well, yes and no. Yes, the X2600 was worth all our efforts. It's hard to tell about the 7700.

6.1.1. X2600

The X2600 prototype was everything that a prototype could be. Unreliable. Flakey. Slow. But... it actually works (sometimes); and when it does work, it works well. At this date, the X2600 has been replaced by the Warlord printer described earlier connected to yet another Alto[7] running a different PRESS formater. This improved the output quality, font catalog and reliability immensely (as predicted).

---

[6] This use of units is *not* to be confused with 18 units per em.

[7] "Old Altos never die, they just become servers"

6.1.2. Editwriter 7700

It was clear to us very early on that the 7700 had the wrong interface for TROFF. Obviously, we would have liked to plug directly into the output device rather than through the Editwriter. Still, we persevered because we thought we could at least get a "hack" solution. Our solution required to us to generate huge files because of the lack of concise and precise position commands. We also realized early on that this would present a space conflict on the Editwriter's floppy. We made some effort to discover the maximum size of a file on the floppy. Our local people considered themselves to be artists and only had vague ideas about this. Reading the manuals didn't help, either. Although the average floppy file will hold a legal size page of normal text, we found that it would only hold a half page from our post-processor. This considerably diminishes its usefulness.

Our life would have been made considerably easier if TROFF had a more flexible view of the output device. It would be nice if TROFF had a way to describe the level of the typesetter. Part of the problem is that TROFF has a *very* low level view of possible typesetters. This, of course, is history, since TI-TROFF is a heavily hacked-up version of the C/A/T typesetter that featured these low level commands.

6.2. Future Recommendations

The output format of TROFF works extremely well in our opinion. It is well designed and easy to interpret. The fact that it is an ASCII file (and readable) makes it relatively easy to debug.

The hand modifications of the input filters demonstrate that something was lacking in the typesetter description file. On one hand, these were mostly hacks to get the output as beautiful as possible, and therefore can be considered "fine tuning". However, the question remains about how to specify such parameters.

Perhaps the greatest area for improvement is the description of both the fonts and the typesetter. In particular, the font descriptions should have have the sizes bound to them and not the typesetter. Furthermore, since faces are standard in fonts, there should be a way to relate fonts to one another (this font is the bold version of ...). As we mentioned above, there should be more flexible ways to create inheritence networks for character searching. There should be some way to describe what the primitive commands of the typesetter are and how they react (a semantics of typesetting?). Also there should be a class and sub-class specification language that can define a device class and then specialize that class for specific instances. And finally, there should be some way to describe all of the measurement units of a typesetter.

7. Acknowledgements

Clearly, we wouldn't have even have started on this project if it hadn't been for Brian Kernighan's labors. We thank him for his efforts and for helping us with the "\|" problem. We only hope that he doesn't take our criticism of TROFF personally.

We owe a large debt to the Xerox Webster Research Center for providing us with the X2600 and later the XP-12 based Warlord. It helped entice people (and addict them) to the glories of typesetting.

Shirley Kelly of Compugraphics withstood my (MWK) questioning for several hours. Her assistance in figuring out what "units" *really* meant were crucial to our "success" with the 7700.

Finally, we are indebted to the folks who have written previous PRESS software. This includes (at CMU) James Gosling (for "cz"), Mike Accetta, Thomas Rodeheffer and Ivor Durham and (at Stanford) Bill Nowicki.

8. Bibliography

[cri65]
P. A. Crisman, ed., *The Compatible Time-Sharing System*, M.I.T. Press, Cambridge, Mass., 1965.

[ker78]
B. W. Kernighan, M. E. Lesk and J. F. Ossanna, UNIX Time-Sharing System: Document Preparation, *Bell Sys. Tech. J. 57*, 6 (1978), 2115-2136.

[ker81]
B. W. Kernighan, A Typesetter independent TROFF, CSTR 97, Bell Laboratories Report, 1981.

[tha81]
C. W. Thacker, E. M. McCreight, B. W. Lampson, B. W. Sproull and D. R. Boggs, Alto: A Personal Computer, in *Computing Structures: Principles and Examples (2nd. Ed)*, D. Sieworek, C. G. Bell and A. Newell (ed.), McGraw-Hill, New York, 1981, 549-582.

APPENDIX A

Handy characters to define

The following table contains a list of characters that should (and in some cases *must)* be defined for TI-TROFF to work properly. This is the voice of experience.

\(hy	-	Character to be used when TROFF hyphenates words. If you don't define it, TROFF will break the word without any indication. (i.e. TI-TROFF won't substitute an ascii "-")
\(em	—	¾ em dash. Used by -me macros
| \^	(1/6 em space) (1/12 em space)	Heavily used in EQN. Not defining it will lead to unpredictable gobbling of succeeding characters. If you define them to have the (un-documented but reserved) typesetter code 0 they will be eaten by the post-processor.
\(eq \(mi \(pl	= - +	These are the equation version of the regular ascii characters =, −, and +. They are invariant w.r.t. to the current font (i.e. one never gets an italics equal sign.) Used heavily by EQN.
\(ru	_	Used for building up horizontal lines. Two adjacent copies of this character shouldn't have any white space between them. Not defining this character will cause a floating point exception (!!!). Used in EQN and lots of other places.
\(br	|	Really a vertical rule that is displaced enough w.r.t. to the current base line so that it mates with the left size of a \(ul to form a seamless corner. Its height at any one time is the current point size. Thus unbroken vertical lines can easily be formed. Used by TBL.
\(sr	√	Obviously needed for EQN. Must have a width exactly the length of the character (no white space on the right).
\(rn		Obviously needed for EQN. Its height and weight must match the square root symbol so that it can be used to extend out square root symbols to arbitrary lengths.
\(ul	_	This is the underline character. It is distinguished from the rule because it is located below the baseline. Used by TBL to form boxes.

APPENDIX B

Special character table

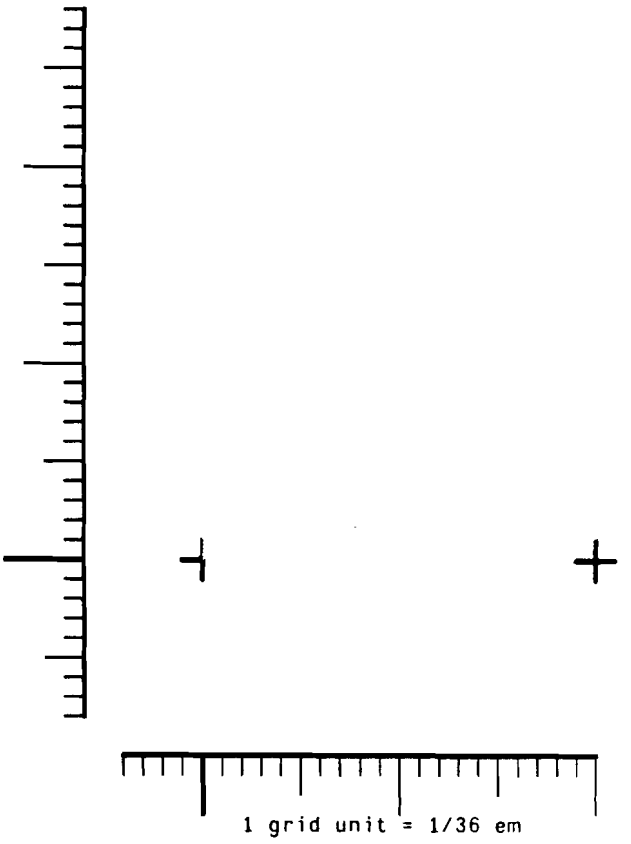
On the following pages are detailed definitions of some of the special characters that we had to build. They are draw on a grid where each unit equals 1/36 em. The baseline is indicated by a dark stroke. The left and right anchors of the characters are indicated by "+"s. When printing, the left anchor is placed at the current position. After the character is imaged, the current position is moved to the location of the right anchor. Any portion of a character that appears to the left of the left anchor is a left kern.

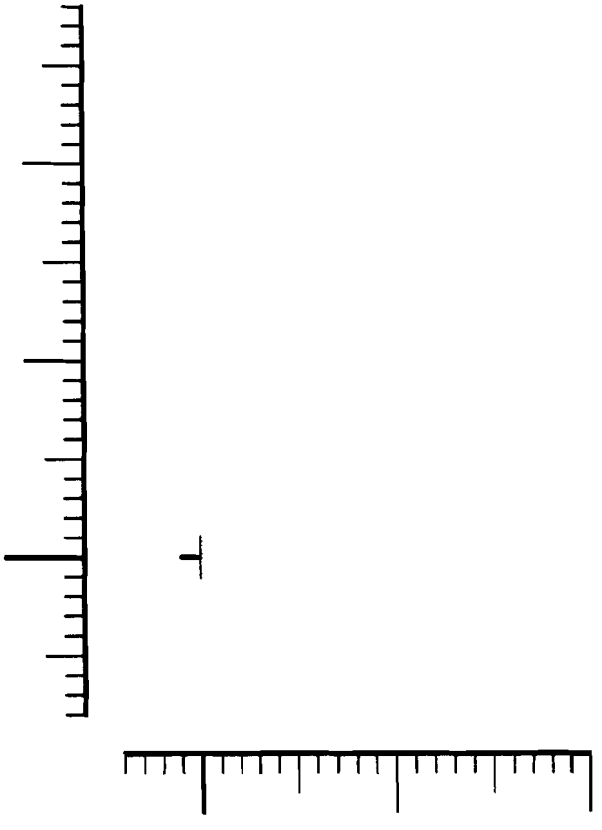
Note that \(bv (bold vertical) is not the same as \(br (box vertical rule). Note also that \(rt, \(rb, \(lk, \(rf and \(rc all have a slight left kern. This makes the text which is to the left of them slightly squeezed. A more aesthetic solution would have been to define another \(bv that was slightly offset. Unfortunately, this would also mean changing pre-processor(s).

Character Name; rule

Troff Name: \(ru

Note that adjacent rule characters join sleamlessly together.





Character Name: box rule

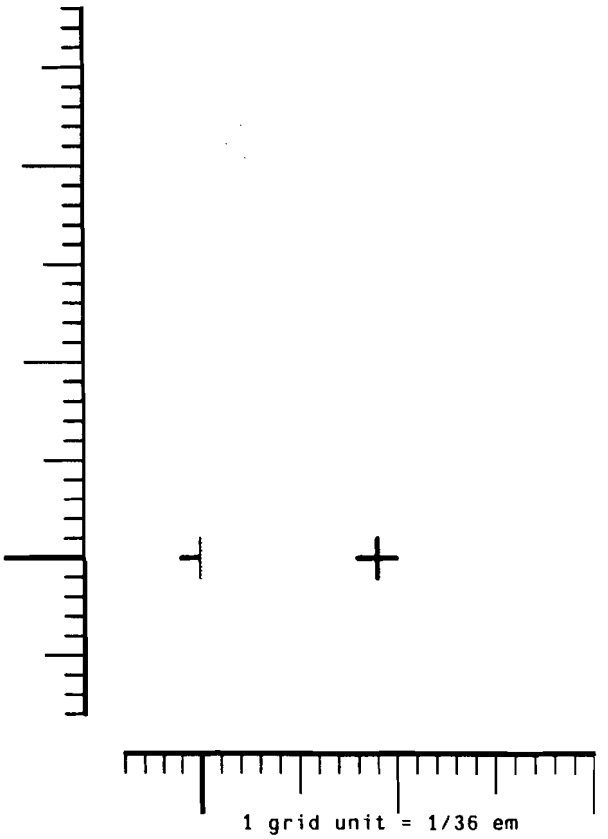
Troff Name: \(br

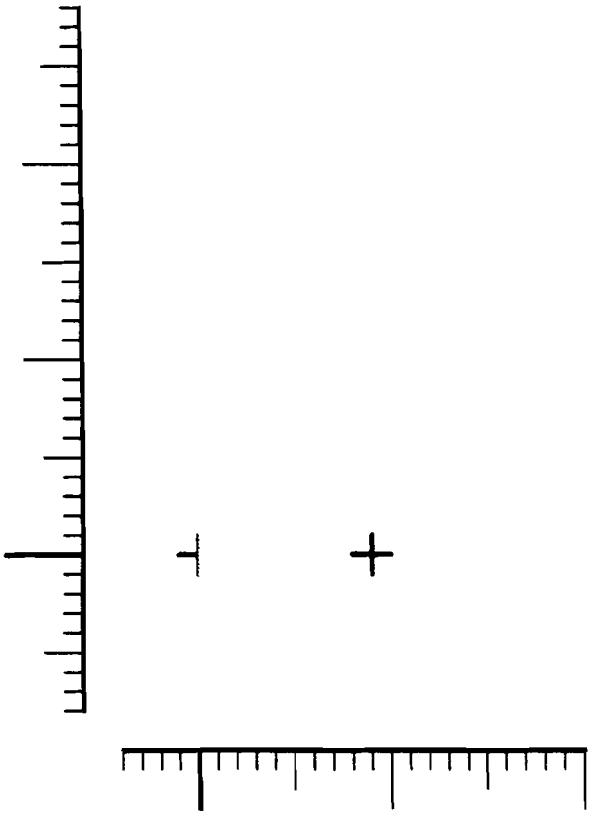
A zero width character

Character Name; left top of big curly bracket

Troff Name: \(lt

Note: all bracket building characters should have the same width.



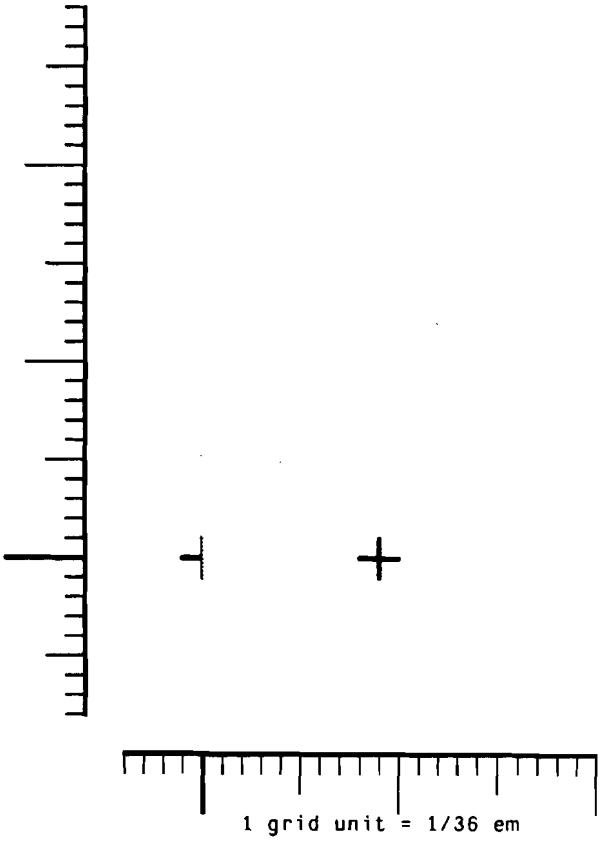


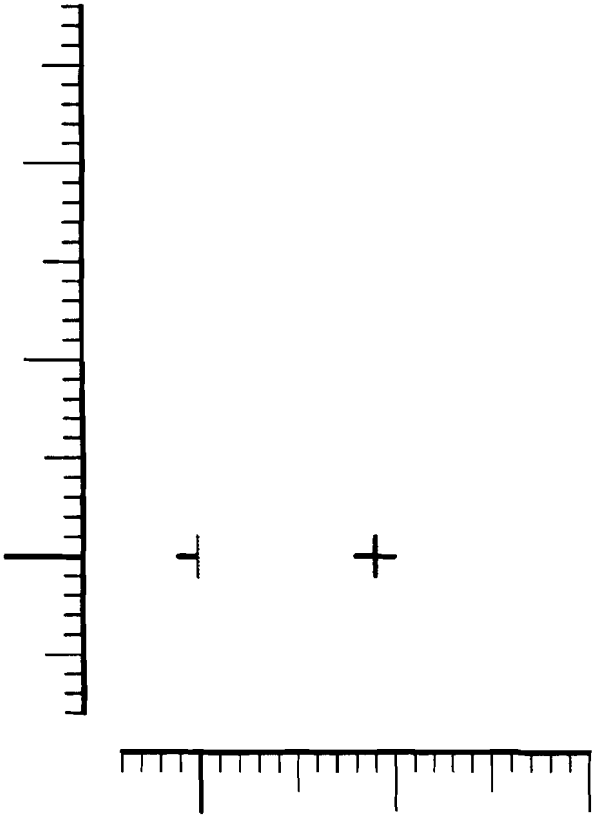
Character Name: left bottom of big curly bracket

Troff Name: \(lb

Character Name; right top of curly bracket

Troff Name: \(rt



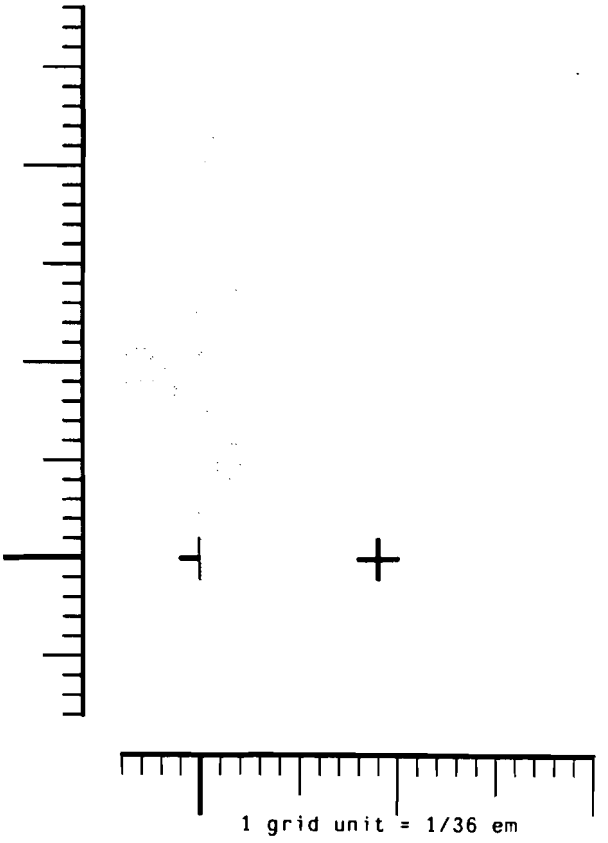


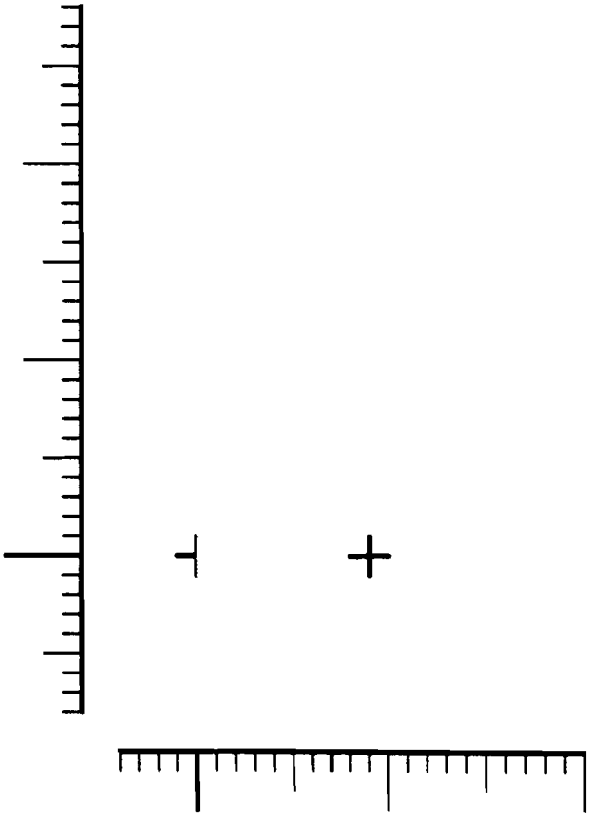
Character Name: right bottom of curly bracket

Troff Name: \(rb

Character Name; left center of big curly bracket

Troff Name: \(lk



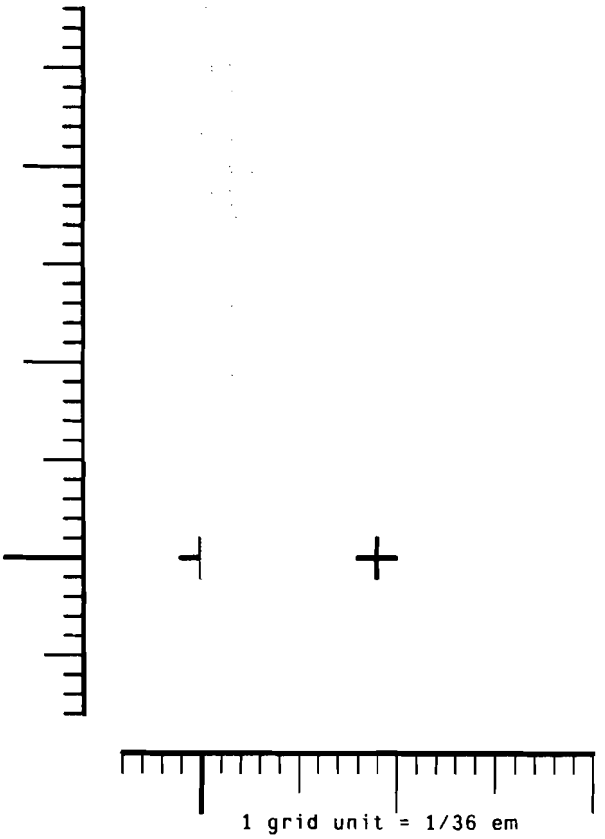


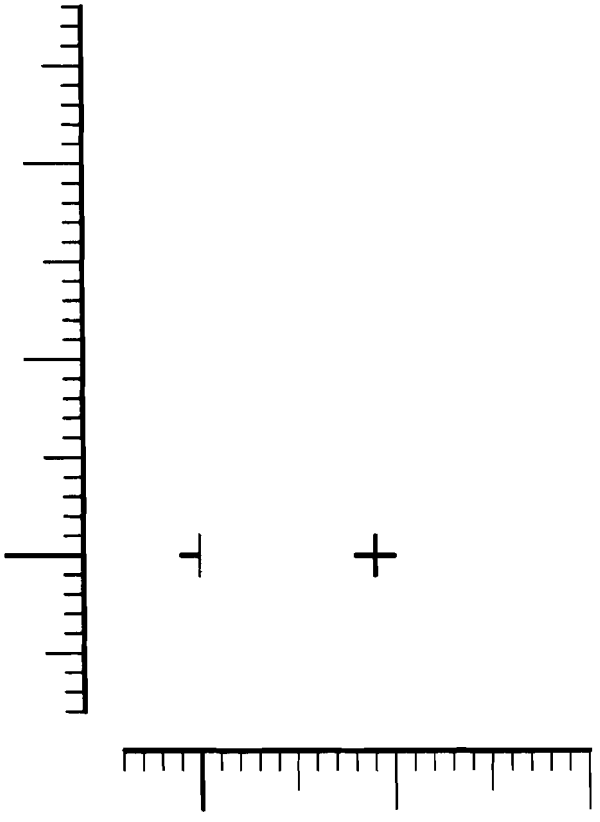
Character Name: right center of big curly bracket

Troff Name: \(rk

Character Name; bold verticle

Troff Name: \(bv



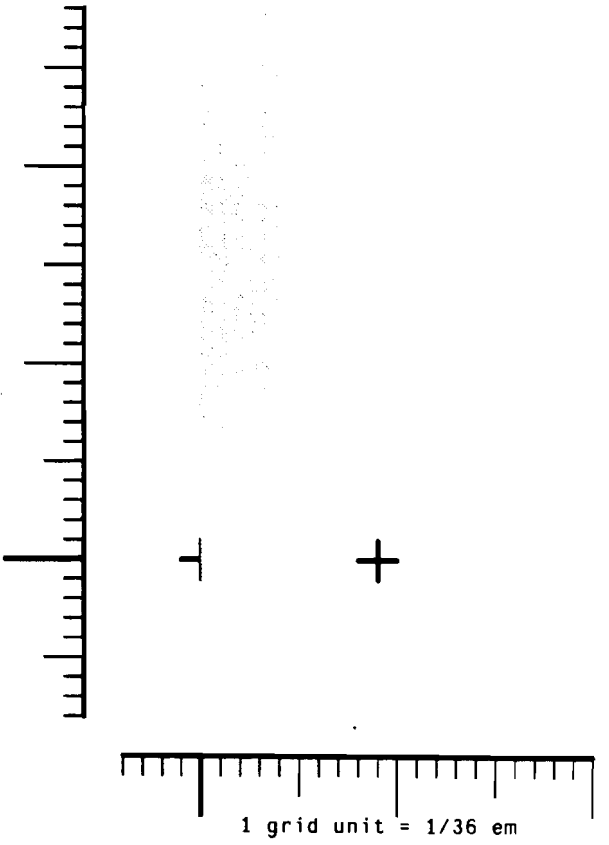


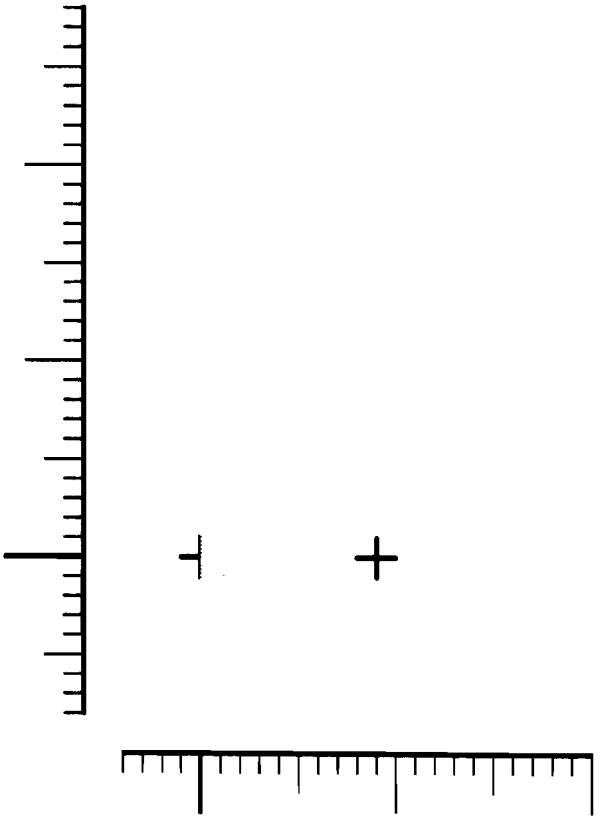
Character Name: left floor
(left bottom of big bracket)

Troff Name: \(lf

Character Name; right floor
(right bottom of big bracket)

Troff Name: \(rf



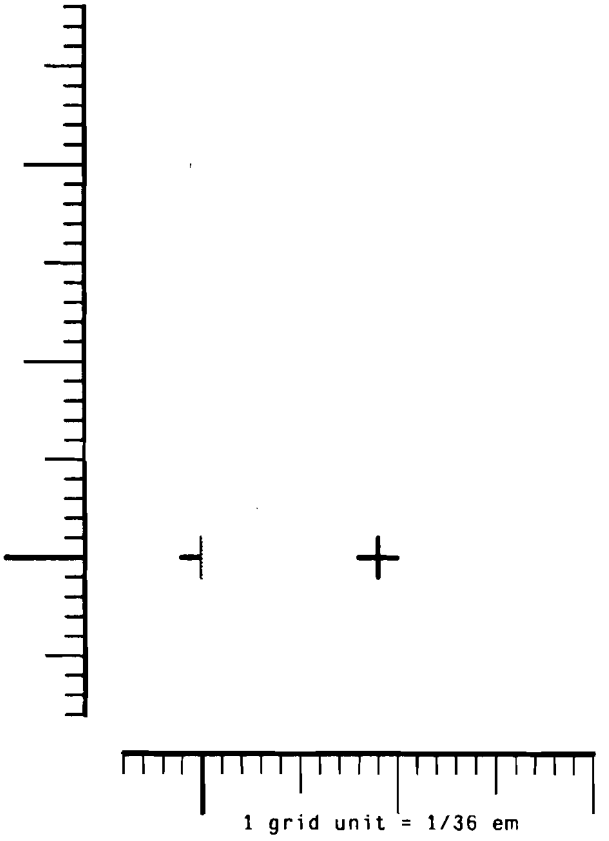


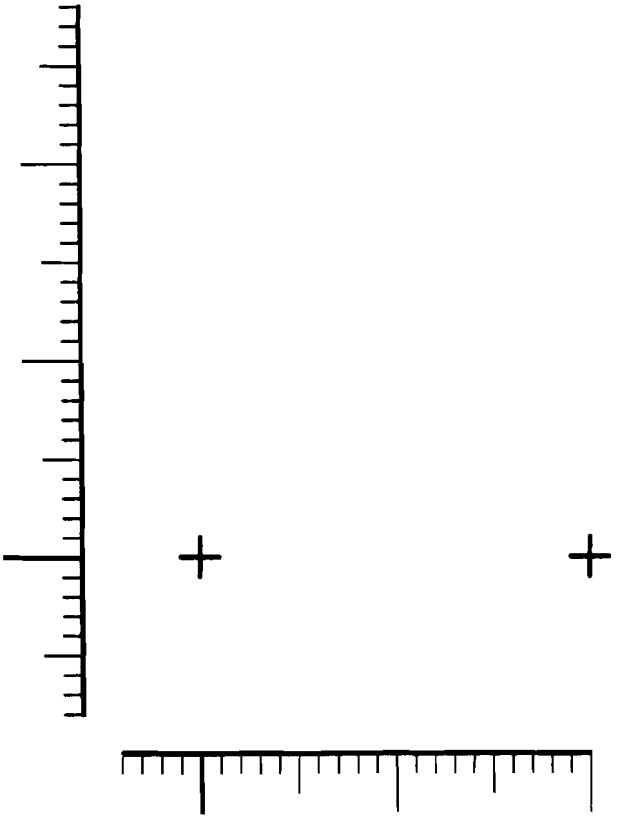
Character Name: left ceiling
(left top of big bracket)

Troff Name: \(lc

Character Name; right ceiling
(right top of big bracket)

Troff Name: \(rc





Character Name: under-rule

Troff Name: \(ul

mates with "box verticle rule" to
create boxes.

APPENDIX C

Special character names in Troff

In troff, special characters are named with one or two character sequences. With the advent of typesetter-independent Troff, implementors have been making up new special character names for symbols that weren't on the Wang (née Graphics Systems) C/A/T typesetter at Bell Labs.

In a effort to help standardize the names of special characters, the following table of known names is provided. It sorted in the ASCII collating sequence. Note that one character names are listed here with a preceding backslash. This is because it is the way they come out in the DESC file. The list has all the old Troff symbols plus others that we have run into.

Because of limitations in our output device, we can't show here a picture of every symbol. In order that we might be un-amiguous, we have included the Xerox character code for most symbols. In this way, we hope that if there is confusion, people will be able to look at the Xerox Character Code Standard (XSIS 058404) and find a pictorial representation.

Characters are tagged with the first implementation that we believed used that name. The assignations are a best-guess and shouldn't be taken too seriously.

name	symbol	XC code	english description	origin
!!	¡	0 | 241	exclaimation down (see also "\(I!")	O
!<			?	A
!<		357 | 104	not less than	X
!=	≠	41 | 142	not equals	C
!>			?	A
!>		357 | 105	not greater than	X
!S	⊅	357 | 136	not super-set	UR
!d		357 | 107	does not divide	X
!m	∉	357 | 113	not a member of (see also "\(nm")	X
!p		357 | 111	not parallel	X
!s	⊄	357 | 137	not sub-set	UR
$D		357 | 242	Dutch Guilder	X
$F		357 | 243	French Franc	X
$J		0 | 245	Japanese Yen	X
''		0 | 315	double acute accent	X
*!	★	41 | 172	five-pointed star	UR
**		356 | 52	math star	C
*A	A	46 | 101	Alpha	
*B	B	46 | 102	Beta	C
*C	Ξ	46 | 121	Xi	C
*D	Δ	46 | 105	Delta	C
*E	E	46 | 106	Epsilon	C
*F	Φ	46 | 132	Phi	C
*G	Γ	46 | 104	Gamma	C
*H	Θ	46 | 113	Theta	C
*I	I	46 | 114	Iota	C
*K	K	46 | 115	Kappa	C

*L	Λ	46 | 116	Lambda	C
*M	Μ	46 | 117	Mu	C
*N	Ν	46 | 120	Nu	C
*O	Ο	46 | 122	Omicron	C
*P	Π	46 | 123	Pi	C
*Q	Ψ	46 | 134	Psi	C
*R	Ρ	46 | 125	Rho	C
*S	Σ	46 | 126	Sigma	C
*T	Τ	46 | 130	Tau	C
*U	Υ	46 | 131	Upsilon	C
*V			???	O
*W	Ω	46 | 135	Omega	C
*X	Χ	46 | 133	Chi	C
*Y	Η	46 | 112	Eta	C
*Z	Ζ	46 | 111	Zeta	C
*a	α	46 | 141	alpha	C
*b	β	46 | 142	beta	C
*c	ξ	46 | 161	xi	C
*d	δ	46 | 145	delta	C
*e	ε	46 | 146	epsilon	C
*f	φ	46 | 172	phi	C
*g	γ	46 | 144	gamma	C
*h	θ	46 | 153	theta	C
*i	ι	46 | 154	iota	C
*k	κ	46 | 155	kappa	C
*l	λ	46 | 156	lambda	C
*m	μ	46 | 157	mu	C
*n	ν	46 | 160	nu	C
*o	ο	46 | 162	omicron	C
*p	π	46 | 163	pi	C
*q	ψ	46 | 174	psi	C
*r	ρ	46 | 165	rho	C
*s	σ	46 | 166	sigma	C
*t	τ	46 | 170	tau	C
*u	υ	46 | 171	upsilon	C
*w	ω	46 | 175	omega	C
*x	χ	46 | 173	chi	C
*y	η	46 | 152	eta	C
*z	ζ	46 | 151	zeta	C
+-	±	0 | 261	plus-minus	C
-+			?	A
-+	∓	351 | 175	minus-plus	NYU
->	→	0 | 256	right pointing arrow	C
-|	⊣	357 | 67	right perpendicular	X
..	¨	0 | 310	diaeresis (or umlaut) (see also "\(um")	M
12	½	0 | 275	1/2	C
13		357 | 375	1/3	X
14	¼	0 | 274	1/4	C
18		357 | 334	1/8	X
23		357 | 376	2/3	X
34	¾	0 | 276	3/4	C
38		0 | 335	3/8	X
4d			?	A

58		0 | 336	5/8	X
78		0 | 337	7/8	X
:>	⇒	357 | 117	double arrow right (see also "\(im")	NYU
<–	←	0 | 254	left pointing arrow	C
<<	≪	357 | 102	much less than	NYU, X
<:		357 | 115	double arrow left	NYU
<=	≤	41 | 145	<= (less than or equal)	C
<>			?	A
<>	↔	357 | 116	double headed arrow (usually differs from "\(io")	X
=.			?	A
==	≡	357 | 162	identically equal	C
=p			?	A
=|		357 | 71	right 2 perpendicular	X
X			?	A
>=	≥	41 | 146	>= (greater than or equal)	C
>>	≫	352 | 103	much greater than	NYU
?=		357 | 164	questioned equality	X
A:	Ä	361 | 47	uppercase A with an umlaut accent	UR
AE		0 | 341	uppercase AE digraph	X
Ao	Å	361 | 41	uppercase A with circle accent	X
C#	ℂ	357 | 254	complex number	UR
CB			?	A
DT			?	A
EG			?	A
EL			?	A
Fi	ffi	360 | 42	ffi ligature	C
Fl	ffl	360 | 43	ffl ligature	C
GE			?	A
I!	¡	0 | 241	inverted ! (used in Spanish) (see also "\(!!")	UR
I?	¿	0 | 277	inverted ? (used in Spanish)	UR
IK		0 | 346	uppercase IJ digraph	X
L.			Small period	M
L.			Large period	NYU
LE			?	A
NE	↗	357 | 76	north-east pointing arrow	UR
NW	↖	357 | 74	north-west pointing arrow	UR
No		357 | 250	number (No.)	X
O/		0 | 351	uppercase O with slash though it	X
O:	Ö	361 | 124	uppercase O with umlaut accent	UR
OE		0 | 252	uppercase OE digraph	X
PL		0 | 250	L w/ slash (Polish L)	O
Pl		0 | 270	l w/ slash (Polish l)	O
QE	▸	357 | 270	Q.E.D.	X
R#	ℝ	357 | 256	Real Numbers	UR
RB			?	A
RC			?	A
SE	↘	357 | 75	south-east pointing arrow	UR
SQ			?	A
SW	↙	357 | 77	south-west pointing arrow	UR
Sl		0 | 154	script l	M, NYU
Sl			?	A
UT			?	A
[[			mark right	NYU

\'	´	357 | 302	acute accent (see also "\(aa")	C
\–	–	0 | 55	current font minus	C
\^			1/12 em half-narrow space character	C
_			rule character	A
\`	`	0 | 301	grave accent (see also "\(ga")	C
\\|			1/6 em narrow space character	C
]]			mark right	NYU
__	⎵		blank space symbol	UR
a+	⊕	357 | 142	abstract plus (circle plus)	X
a-	⊖	357 | 143	abstract minus (circle minus)	X
a/	⊘	357 | 145	abstract divide (circle divide)	X
a:	ä	361 | 247	lowercase a with umlaut accent	UR
a^		?	lower-case circu accent	CWI
aa	´	0 | 301	acute accent (see also \')	C
ab		?	lower-case diaer accent	CWI
ad		?	lower-case breve accent	CWI
ae		0 | 261	lower-case ae diph	CWI
ag	∠	357 | 154	angle	X
ai			?	O
al	ℵ	357 | 247	generic infinity (aleph)	M,NYU
an			?	A
an	∧	357 | 266	boolean "and"	NYU
ao	å	361 | 250	lowercase a with circle accent	UR
ap	~	356 | 166	approximates (middle level tilde)	C
as			?	A
ax	⊗	357 | 144	abstract multiply (circle-times)	X
b0			45 degree tilt	NYU
b9			90 degree tilt	NYU
bc			?	A
bc	∵	357 | 157	three dots meaning: because	X
be	˘	0 | 306	breve accent	X
br	|	357 | 344	box vertical rule	C
bs			Bell System logo	C
bt	⊥	357 | 160	bottom (perpendicular)	NYU
bu	•	357 | 146	bullet	C
bv	⎪	356 | 341	bold vertical (middle part of a big bracket)	C
bx	▮	42 | 43	black box	M,A,NYU
c,	ç	361 | 255	lower case c with cedilla	UR
ca	∩	357 | 126	cap (set intersection)	C
cd		0 | 313	cedilla	M,NYU
cf	%	357 | 100	in care of, c/o	X
ci	○	41 | 173	big open circle	C
cm	∋	357 | 114	contains as a member (mirror image of "\(mo")	X
co	©	0 | 323	copyright symbol ("c" in a circle)	C
cp			cap (upside down breve)	O
cr	⤶	360 | 275	printing symbol for carriage return	UR
cs	∲	357 | 166	contour integral sign	X
ct	¢	0 | 242	cent sign	C
cu	∪	357 | 127	cup (set union)	C
cy			?	A
d<		0 | 253	double left guillemet	X
d>		0 | 273	double right guillemet	X
da	↓	0 | 257	down arrow	C

dc			?	O
dd	‡	357 | 61	double dagger	C
de	°	0 | 260	degree	C
dg	†	357 | 60	dagger	C
di	÷	0 | 270	divide	C
dm	◇	41 | 176	diamond	NYU
dt		0 | 307	dot accent	X
eg			?	A
el			?	A
em	—		3/4 em dash	C
en			? (en-dash? 3/4 en-dash?)	M
eq	=	0 | 75	math equals (invarient w.r.t. current font)	C
es	∅	357 | 171	empty set	C
fa	∀	357 | 265	for all (inverted A)	NYU
fe		41 | 152	female	M,NYU
ff	ff	360 | 41	ff ligature	C
fi	fi	360 | 44	fi ligature	C
fl	fl	360 | 45	fl ligature	C
fm	'		footmark	C
ga	`	0 | 301	grave accent (see also `)	C
ge			?	A
gp			?	A
gr	∇	0 | 357 | 271	gradient (nabla)	C
h/		357 | 150	Plank's constant	UR
hc	ˇ	0 | 317	hacek (caron)	M, NYU
hy	-	41 | 76	hyphen	C
ib	⊆	357 | 131	improper subset	C
if	∞	41 | 147	infinity	C
ij		0 | 366	lowercase ij digraph	X
im	⇒	357 | 117	implies	NYU
io	↔	356 | 116	if and only iff (<=>)	NYU
ip	⊇	357 | 136	improper superset	C
is	∫	357 | 165	integral sign	C
l.			small period	M, NYU
lK	<	357 | 62	left bracket	X
la		357 | 266	logic and (see also "\(an")	NYU
lb	⎩	356 | 345	left bottom of big curly bracket	C
lc	⎡	356 | 163	left ceiling (left top of big square bracket)	C
ld			?	A
le			?	A
lf	⎣	356 | 164	left floor (left bottom of big square bracket)	C
lh		357 | 265	left pointing hand	C
li		357 | 265	litre (liter)	X
lk	⎨	356 | 346	left center of big curly bracket	C
lo	∨	357 | 267	boolean "or" (\/)	NYU
lq		0 | 252	double left quote	NYU
lt	⎧	356 | 344	left top of big curly bracket	C
m.		0 | 267	math dot (centered dot)	NYU
ma		41 | 151	male	M, NYU
ma	¯	0 | 305	macron accent X	
mc	∘	357 | 147	math composition (centered small circle)	NYU
mc		0 | 55	macron (see also "\(ma")	O
mi	–	0 | 55	math minus (invarient w.r.t. current font)	C

mo	∈	357 | 112	member of	C
mt		41 | 154	minutes	X
mu	×	0 | 264	multiply symbol	C
n'		357 | 47	neutral single quote	X
nm	∉	357 | 113	not member (see also "\(!m")	NYU
no	¬	357 | 152	"not" symbol	C
o/		0 | 371	lowercase o with slash through it	X
o:	ö	361 | 324	lowercase o with umlaut accent	UR
ob			open circle (outline bullet)	M
ob			?	A
oe		0 | 372	lower-case oe diph	CWI, X
og		0 | 316	ognek accent	X
or	|	357 | 106	"or" bar	C
os			?	A
pc		357 | 124	properly contains, type 1	X
pd	∂	357 | 272	partial derivative	C
pi		357 | 125	properly contained in, type 1	X
pl	+	0 | 53	math plus (invarient w.r.t. current font)	C
pm		357 | 101	per mil, %o	X
po	£	0 | 243	pound sterling (see also "\(ps")	NYU
pp	¶	0 | 266	paragraph symbol	M, A, NYU
ps	£	0 | 243	pound sterling (see also "\(po")	O
ps			?	A
pt	∝	357 | 161	proportional to	C
r1	⇋	357 | 120	reversible reaction, type1	X
r2	⇄	357 | 121	reversible reaction, type2	X
r>	>	357 | 63	right angle bracket (not less than)	X
ra			?	A
rb	⌋	356 | 343	right bottom of big curly bracket	C
rc	⌉	356 | 166	right ceiling (right top of big square bracket)	C
rd			?	A
rf	⌋	356 | 167	right floor (right bottom of big square bracket)	C
rg	®	0 | 322	registered symbol ("r" in a circle)	C
rh		357 | 64	right pointing hand	C
ri	°	0 | 312	ring (or circle) accent	X
rk	⎭	238 | 342	right center of big curly bracket	C
rn		257 | 320	root en extender (matches top of "\(sq")	C
rq		0 | 272	right double quote	NYU
rt	⌉	238 | 340	right top of big curly bracket	C
ru	_	?	baseline rule	C
sa			?	O
sb	⊂	357 | 133	subset of (right horseshoe)	C
sc	§	0 | 247	section marker	C
sd		41 | 155	seconds	X
sg		357 | 155	spherical angle	
sl	/	0 | 57	slash (matching backslash)	C
sp	⊃	357 | 132	superset of (left horseshoe)	C
sq	□	42 | 42	square	C
sr	√	357 | 174	square root	C
ss		0 | 373	german script ss	M
st	⊢	357 | 66	turnstyle (see also "\(|-")	NYU?
te	∃	357 | 264	there exists (reflected E)	NYU, UR
tf			?	A

tf	∴	41 | 150	three dots meaning: therefore	X
tm		0 | 324	trademark	X
to		357 | 160	top (inverted bottom) (see "\(bt")	NYU
tp		357 | 274	telephone	M, NYU
tr	△	42 | 44	triangle	UR
ts	ς	46 | 167	terminal sigma	C
u:	ü	361 | 245	lowercase u with umlaut accent	UR
ua	↑	0 | 255	up-arrow	C
ud	.	43 | 326	under (low) dot accent	UR
ui		0 | 365	undotted i	NYU
uj		o | 345	undotted j	NYU
ul	_	0 | 137	under-rule (mates with "\(br")	C
um	¨	0 | 310	umlaut (diaeresis) accent (see also "\(..")	X
vr		?	verticle rule - lines up with \(ru	M
wi			?	O
yi		0 | 245	Japanese yen (see also "\($J")	O
|–	⊢	357 | 66	turnstyle (see also "\(st")	X
|=	⊨	357 | 70	left 2 perpendicular	X
||	∥	41 | 102	is parallel	UR, X
~=	≃	357 | 167	approximately equals	C
~>	⇝	357 | 123	wiggly arrow	X
~~	≈	357 | 171	squiggly equals	M

Sources:

C the original troff implementation at Bell Labs for the C/A/T phototypsetter.

A AT&T Bell Labs. implementation for the APS-5.

M AT&T Bell Labs. implementation for the Mergenthaler Linotron 202

CWI from paper by Jaap Akkerhuis at European Unix(tm) Systems User Group Autumn Meeting 7th-9th Sept. 1983, Trinity College, Dublin

NYU Post-processor developed by Lou Salkind at NYU for the Imagen.

O Other. Source location could not be determined.

UR University of Rochester -- as described herein

X Post-processor for INTERPRESS developed at the Xerox Webster Research Center.