EXTENDING GNU TROFF TO PRODUCE HTML THROUGH THE TECHNIQUE OF NEXT EVENT SIMULATION

GAIUS MULLEY and WERNER LEMBERG

School of Computing, University of Glamorgan, CF37 1DL, UK
E-Mail: gaius@glam.ac.uk
Kl. Beurhausstr. 1, 44137 Dortmund, Germany
E-Mail: wl@gnu.org

Abstract: This paper reports on a technique used to generate accurate HTML output from GNU Troff. GNU Troff is a typesetting package which reads plain text mixed with formatting commands and produces formatted output. It supports a number of devices and now supports the production of HTML. The paper discusses the design of the HTML device driver `grohtml` and modifications made to GNU Troff. The front end program `troff` was modified to maintain a reduced state machine which is examined each time a glyph is passed to the back end device driver (`post-grohtml`). Any change in system state between the production of two glyphs results in a sequence of events being passed to the device driver. There is a direct correspondence between this technique and creating a script for a next event simulation queue. Furthermore the device driver reconstructs the system state and formats the HTML according to state changes caused when processing the event queue. This technique works well, as it minimises the state information passed from front end to back end device driver whilst still preserving the high level layout of the text. Using this technique GNU Troff effectively translates input source into another mark-up language and thus this technique could be extended to translate GNU Troff documents into any of the OpenOffice supported formats. Troff has been in use for three decades now and is still actively used by authors. Troff's biggest use, however, is to format manual pages for GNU/Linux and other UNIX like operating systems. Introducing this facility into GNU Troff provides users with the ability to translate legacy documents into HTML and in the future to a format supported by OpenOffice.

Keywords: groff, troff, grohtml, HTML, simulation.

INTRODUCTION

GNU Troff is a reimplementation of the program `troff` which is available on the UNIX operating system. The original `troff` was written in PDP-11 assembly language by Joe Ossanna in 1973. Two years later it was rewritten in C and afterwards it went through a series of revisions until Joe Ossanna died in 1977.

Brian Kernighan continued `troff` development for the next 15 years and it is testament to the original design that the input language remained the same. Much of the original documentation received minor changes as new `troff` releases were issued and therefore Joe's name was retained on these manuals [Ossanna, 1992].

Brian Kernighan modified `troff` so that it could produce output for a number of different typesetting devices, while at the same time retaining the same input language specification. The input language was so robust that a number of preprocessors were written to provide reference [Tuthill, 1986], table [Lesk, 1976a], picture [Kernighan, 1991, Wyk, 1982] and equation handling [Kernighan, 1977, Kernighan, 1976] (`refer`, `tbl`, `pic` and `eqn` respectively). These programs were executed in a pipeline and `troff` transformed the heavily preprocessed formatting commands and text into device dependent output [Kernighan, 1978]. Although the input to `troff` was device independent it was also very low level and therefore users were encouraged to use macro sets when producing documents [Lesk, 1976b]. During the 1980s the internals were modestly revised and a number of new macro sets were written [Allman, 1980, Smith, 1980]. The macro sets provided freedom in document styling (similar to modern HTML style sheets) and they included: arbitrary style headers and footers; arbitrary style footnotes; automatic sequence numbering for paragraphs, sections, etc; multiple column output; dynamic font and point-size control; arbitrary horizontal and vertical local motions at any point; mathematical bracket construction, and line drawing functions [Ossanna, 1992].

During the 1980s `troff` was extended to handle many different devices, this was accomplished by splitting the task of `troff` into two components. The front end `ditroff` produced device independent code and the back end device driver which simply translated the device independent code into the target device commands.

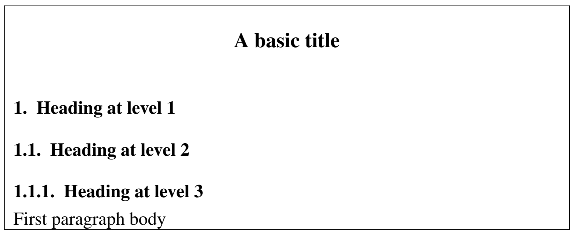
James Clark began to work on the GNU implementation of the `troff` family of tools in 1989. This was to be a completely new implementation of all the preprocessors, the `ditroff` program and the macro sets. The first release of `groff` (version 0.3.1) occurred in June 1990 and it included a replacement for `ditroff`, `eqn`, `tbl` and `pic`. It also included a replacement for the `me` macros and the `man` macros. The replacement programs were mostly written in C++ and often supported extensions and removed various static data size limitations. Since 1999 the `groff` package has had new maintainers and has undergone active development. It supports the common macro sets associated with `troff` (`man`, `me`, `ms`, `mdoc` and `mm`). However `groff` also provides a modern macro set (`mom`) and also provides new preprocessors and support for colour.

DESCRIPTION OF THE PROBLEM

Troff was widely used in the 1970s and 1980s. All UNIX documentation, release notes for both the AT&T and BSD UNIX variants was written in `troff`. Many papers in the CACM and Software Practice and Experience journals were also produced using `troff`. Even today all UNIX and GNU/Linux manual pages are written in `troff` using the `man` macro set and some authors advocate using `troff` above other WYSIWYG tools to typeset their books [Tanenbaum, 1997, Stevens, 1998, Schaffter, 2004].

Groff provides compatibility with `troff` as well as many modern enhancements, image handling, colour and limited pdf mark capability. Clearly the addition of an HTML device driver would be useful.

GNU Troff copied the original design of UNIX Troff , which maps the input source onto a physical device through the use of a device independent language. Through this language it effectively plots each glyph at Cartesian coordinate position on a page. The difficulty in translating `troff` source into HTML is exacerbated by the fact that both HTML and `troff` source are mark-up languages. As the GNU Troff and original UNIX Troff packages rely heavily on the pipeline principle it has naturally led to the preprocessors (`pic`, `tbl` and `eqn`) translating their high level commands onto much lower level `troff` commands. In turn, this has meant that much of the high level information (for example where a table starts and ends) is lost. Furthermore, using any of the macro packages will result in many low level commands to format abstracts, titles, footnotes, paragraphs, lists, hanging indents etc. Consequently, translating `troff` input source into HTML cannot be achieved by writing a device driver which simply reads `ditroff` input and produces HTML output. Consider the diagram in figure 1 which shows the key components of a simple `groff` command line invocation together with a synopsis of the data passing though the pipeline. In figure 1 the `troff` input consists of requests or calls to macros (lines prefixed with a period) and text. Troff input might also include text with escapes, for example the word `program` can be typeset by temporarily reducing the point size by 1 and altering the font to Courier. The word and escapes can be encoded as `\s-1\fCprogram\s+1\fP`. In figure 1 the final output is PostScript (the default device) and the invocation also includes the `ms` macro package. Figure 2 shows how a PostScript printer interprets the output from Figure 1.


A basic title

1. Heading at level 1

1.1. Heading at level 2

1.1.1. Heading at level 3

First paragraph body

Figure 2: displaying the PostScript output

Notice how the `ditroff` output only knows which font is to be used, the font size and position that the glyph should be placed. Another example that an HTML device driver must translate accurately is shown in figure 3.

```
The start of an indented paragraph example
in which line 1, line 2 and line 3 are
vertically aligned.
.LP
.IP once
line 1
.IP twice
line 2
.IP threefold
line 3
```

Figure 3: examples of indented paragraphs

Here we see from the output shown in figure 4 that the `ms` macro set diverted the indented paragraph label parameter *once*, *twice* and *threefold* into a macro. It then tests to see whether the macro width is greater than a default length and if so it breaks the line before starting the indented paragraph. Clearly the HTML device driver needs to cope with these constructs, as they are also heavily used in manual pages.

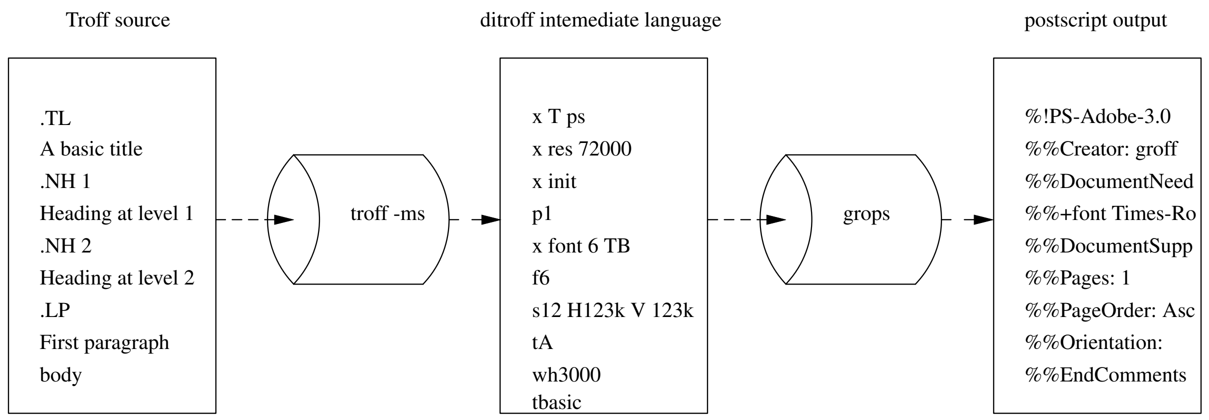
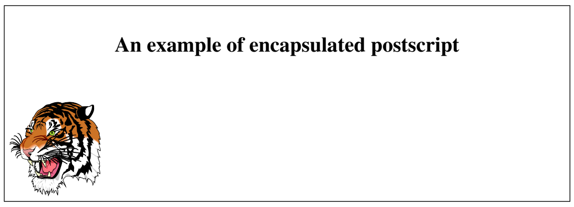


Figure 1: Simple title, heading and paragraph

```
The start of an indented paragraph example in which line 1, line 2 and line
3 are vertically aligned.
once  line 1
twice
       line 2
threefold
       line 3
```
Figure 4: result of processing figure 3 with `groff -ms`

Furthermore many documents will include encapsulated PostScript as shown in figure 5. The processed output is shown in figure 6. The issue here is that PostScript has become the default device driver and groff allows encapsulated PostScript to be handled as a special device case. Nevertheless this feature is so useful and it would be expected to be implemented in the HTML device driver. Groff also allows users to embed their own PostScript inside Troff source files (a technique similar to inlining assembly language within a high level language). In both cases users could reasonably expect the HTML device driver to translate the PostScript into a PNG image which is referenced by the HTML output.

```
.TL
An example of encapsulated PostScript
.LP
.PSPIC -L tiger.eps 1i 1i
```
Figure 5: example of encapsulated PostScript usage



An example of encapsulated postscript

Figure 6: result of processing figure 5 with `groff -ms`

A further problem which needs to be addressed is the limited number of glyphs available in HTML. For example HTML has a restricted set of mathematical glyphs and it is restricted in its ability to accurately position these glyphs [Musciano, 1998]. Groff in contrast can be used to typeset mathematical formulae and users will at least expect the groff HTML device driver to reproduce the formulae as an image. Figure 7 shows Troff input, which describes the definite integral.

```
.EQ
delim $$
.EN
.LP
As $ delta x -> 0 $ then
.EQ
sum from x=a to b y . delta x ~ approx ~ \
int from a to b y ~ dx
.EN
```
Figure 7: `eqn` example

When processed with the command `groff -e -ms` it produces the output as shown in figure 8.

As $\delta x \to 0$ then

$$\sum_{x=a}^{b} y . \delta x \approx \int_{a}^{b} y \, dx$$

Figure 8: output of figure 7 as rendered on a PostScript device

Finally the `troff` language is very rich in commands and requests. The language has the ability to construct macros at run time and it does this by diverting text and commands into a *diversion*. Some authors of macro sets perform some highly impressive programming tricks using this technique. For example this technique has been used in the `ms` macro set to handle the `IP` request and the `TL` request seen earlier. It is also used to manage abstracts, authors names and references for papers. The problem is that when `troff` generates the `ditroff` output the meaning of the text is lost. The HTML device driver needs to know whether a line was centred, if the line was indented, when the centring ends, when indentation ends, when text is a section heading and when text is a title. It cannot guess this reliably by examining the glyph position and font information.

Conveying the indentation information to the device driver is particularly difficult as diversions can be created to manipulate this value when the diversion is re-read. Consider the example given in figure 9.

```
.LP
.di foo
.br
.in +1i
hello
.in -1i
.br
.di
.in +2i
.br
.foo
.in +1i
.br
.foo
```

Figure 9: diversion and indentation example

When processed with the command `groff -ms` it produces the output as shown in figure 9.

```
                                        hello
                                                hello
```

Figure 10: output of figure 9 as rendered on a PostScript device

Clearly an HTML device driver must also understand the diversions sufficiently so that the important formatting requests are accurately translated.

PREVIOUS WORK

There have been a number of scripting solutions implemented which translate `troff` documents into HTML. These scripts operate at the macro set level and they independently translate the functionality of the various macros into HTML. For example scripts exists to translate the `ms` macros `IP`, `TL`, `LP`, `PP`, `DS` into indented paragraph; title; left aligned paragraph; paragraph and block text. The advantage of this approach is its simplicity but the disadvantages are that the scripts need to be maintained independently to the corresponding macro set and more importantly the scripts do not understand user defined macros. Often these scripts do not honour all the macro optional arguments or allow inclusion of encapsulated PostScript.

Another solution is `unroff` [Laumann, 1996], which is a Scheme-based, programmable and extensible `troff` translator which translates `troff` files into HTML-2.0. It supports the `me`, `ms`, and `man` macro sets as well as the front end preprocessors `tbl` and `eqn`. This is an ambitious project and it has been successful in translating manual pages and `troff` documents. The success is partly due to `unroff` understanding many of the underlying `troff` requests and having extra tag information inside its macro set descriptions. The main disadvantages to `unroff` are that it does not have a seamless interface to the `pic` preprocessor and currently it does not handle encapsulated PostScript or diversions.

The `doclifter` project [Raymond, 2005a] provides a means whereby many documents written in `mm`, `ms`, `me` and `mdoc` can be translated into XML-Docbook [Walsh, 1999]. Eric Raymond designed `doclifter` to operate at the semantic level and he acknowledges that to do a really good job requires some human polishing. Nevertheless `doclifter` has successfully translated 95% of all manual pages in Fedora Core 3.

Doclifter translates `tbl` tables into DocBook table markup and `pic` input into `SVG`, however `eqn` is not translated.

OVERVIEW OF GROHTML

The approach taken by the authors in the design of `grohtml` was to modify the front end `groff` binary so that when the HTML device is requested the `troff` input is run through the PostScript device driver and later through the HTML device driver. All images are created by converting the relevant PostScript page into a bitmap image after which the required area is cut and converted into the PNG [Roelofs, 1999] format. The pipeline structure is outlined in figure 11. The preprocessors `tbl`, `eqn` and `pic` were modified to indicate where the start and end of a table, equation and picture occur. This information is then transparently read by the front end and determines the area of the bitmap file which is required for an image. The advantage of this approach is that `grohtml` can fully support encapsulated PostScript inclusion and it also allows users to use this feature together with any combination of glyphs to generate web images.

The macro sets `ms` and `man` were also slightly modified to issue a tag when `troff` encounters a section heading, title or indented paragraph. All tags are emitted together with the `ditroff` intermediate code to the device driver `post-grohtml`.

The `troff` program was also modified to handle another escape `\O` which has five variants and they are described in table 1. These escapes are

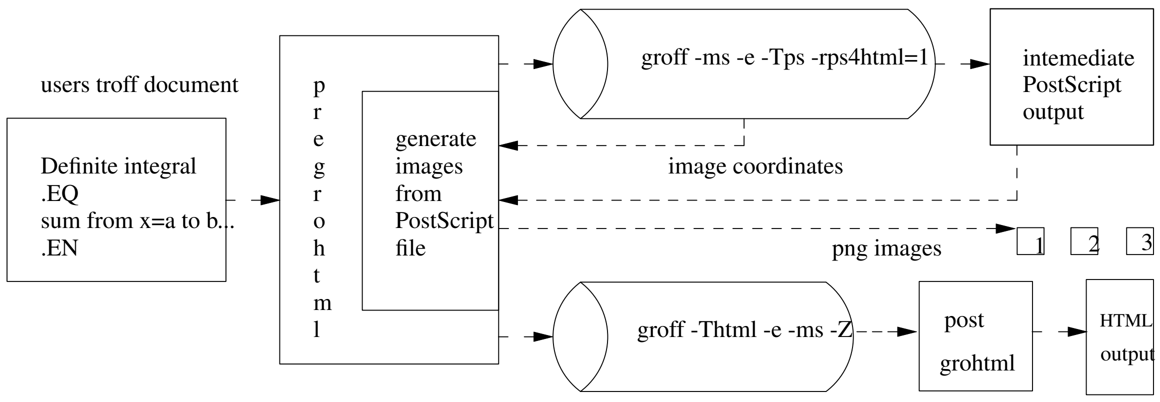


Figure 11: the `groff -Thtml -ms -e` pipe line

Escape	Meaning
`\O0`	Disable any glyphs from being emitted at the outer level. †
`\O1`	Enable output of glyphs, provided that the escape occurs at the outer level. †
`\O2`	Providing the escape occurs at the outer level, enable output of glyphs and also write out to `stderr` the page number and four registers encompassing the glyphs previously written since the last call to `\O`.
`\O3`	Begin a nesting level.
`\O4`	End a nesting level.
`\O[5`*P filename*`]`	HTML device specific. Provided that this escape occurs at the outer nesting level write the *filename* to `stderr`. The position of the image, *P*, must be specified and must be one of `l`, `r`, `c` or `i` (left, right, centre, inline).

Table 1: new `troff` escapes and their meaning

important as they allow preprocessors to tell `troff` to skip a block of text or issue a box of coordinates which define the area for a set of glyphs. This functionality is used during the PostScript pass to determine the region used by a table, picture or equation. Conversely it is used by the HTML pass to stop producing glyphs when a table, equation or picture is encountered; but replace these items with an HTML reference to an image.

STATE MACHINE SOLUTION

If `grohtml` is implemented as described in the last section it will be able to translate all: encapsulated PostScript, `tbl`, `pic`, `eqn`, section headings, numbered headings and titles accurately. However it will still fail to accurately translate: centred lines, any indentation, preformatted blocks, tabulated text, line breaks and vertical spaces. A solution to this problem is to extend the repertoire of tags which are passed from `troff` to `post-grohtml`. This requires further modification to the `troff` so that it will issue tags whenever the above formatting requests are encountered. In essence the back end of `troff` maintains a simple state machine which represents the minimum information required to correctly reproduce the formatting requests listed above. Whenever a glyph is about to be written in the `ditroff` output, `troff` needs to check whether this state has changed since the last glyph was written. If the state has changed then `troff` emits a series of tags which define the change in state. There is clear correspondence between tags appearing in the `ditroff` intermediate output and a next event queue albeit without any time component. The HTML driver `post-grohtml` reconstructs the system state according to the events it reads and in the process translates the glyphs into HTML text surrounded by the appropriate HTML tags [Musciano, 1998]. Furthermore `troff` can handle requests read from diversions by pushing the current minimal HTML state before a diversion is read and popping it later once the diversion has finished. This allows diversions to centre lines, alter the indentation, and emit preformatted text, whilst restoring the state once the diversion has ended.

The complete set of tags which post-grohtml understands is shown in table 2. Figure 16 shows the extended ditroff output when GNU Troff processes Figure 9. The command line to generate this

† Both `\O0` and `\O1` reset four `troff` registers which mark the top left and bottom right hand corners of an area containing all written glyphs.

information is shown below:

```
groff -Z -ms -Thtml ex9.n
```

and the output in Figure 16 contains the minimum tags necessary in order to convey the meaning of the glyph formatting. The example shown in figure 9 produces two lines with a single word `hello` on each line. The first line `hello` is indented at three inches and on the second line `hello` is indented at four inches. The example starts by declaring a left adjusted paragraph `.LP` and then it creates the diversion `.di foo`. All subsequent text and requests are diverted into `foo` until the second `.di` is encountered. The next request `.in +2i` informs the typesetter that an indentation of two inches is required. When `.foo` is encountered it rereads the diversion which performs another relative indent of one inch before emitting the glyph `hello`. The diversion then reduces the indentation by one inch before finishing. Textual input is resumed after the diversion `.foo` at the request `.in +1i` which performs a relative indent of one inch (at this point the indentation is set at three inches). Lastly the diversion is run again which emits the final `hello` four inches from the left hand margin.

Figure 16 shows the ditroff output together with an annotated description. It also shows extensions made to ditroff which include the extra tag information (lines starting with `x X devtag`: convey the minimum events necessary such that `post-grohtml` can reconstruct the `troff` state machine. As `post-grohtml` reconstructs the state machine it generates the appropriate HTML. Critically for this example it can be seen that here are only two `x X devtag:.in` events, one at 3 inches and the second at 4 inches, despite the example in figure 9 having four `.in` requests.

EXAMPLES OF GROHTML OUTPUT

The examples shown in this section were produced with `groff` version `1.19.2` and the test cases are those from figures 3, 5, 7 and 9. All the examples were constructed using the following pipeline:

```
groff -Thtml -e -ms filename.ms | html2ps
```

and these are shown in figures 12, 13, 14 and 15 respectively. The test results appear satisfactory and show that the indentation using the `IP` macro works as expected. This macro uses a diversion to detect the width of its first argument before emitting the paragraph. The example shows how each line 1, line 2 and line 3 are vertically aligned. Notice that line 3 has a vertical space between itself and `threefold`. Figure 13 shows that `groff` can translate `eqn` input into HTML and images and figure 15 shows that indentation within diversions behave correctly.

At present the GNU Troff macro sets: `ms`, `man`, and `mwww` have the extra HTML tags included. Figure 17 shows the output of a component from the GNU `pic` guide [Raymond, 2005b] when processed through the following pipeline.

```
groff -pet -ms -Thtml pic.ms | html2ps -s2
```

It can be seen that `grohtml` correctly determines that the equation is inside a picture and generates one image. Figure 18 shows the `grohtml` manual page after it was processed by the pipeline:

```
groff -pet -man -Thtml grohtml.man
```

and viewed through a browser. In this case we can clearly see the vertical alignment correctly represented between lines which contain a command line option and those lines which are a component of a hanging paragraph. Figure 19 shows how the GNU Troff homepage appears when printed from a browser. The homepage was created by the following command line.

```
groff -pet -ms -Thtml webpage.ms
```

The file `webpage.ms` make extensive use of the `mwww` macro set which provides drop capital, unordered line list, URL and image related macros. It also makes use of colour which was recently added to GNU Troff.

CONCLUSIONS AND FURTHER WORK

In conclusion the `grohtml` device driver has been constructed and integrated into `groff`. This work has taken six years of part time effort and amounts to approximately 9600 lines of C++ code (not including macro set modifications). In due course the `groff-1.19.2` will be formally released and this will be the first beta release of `grohtml`. During the development of `grohtml` a number of additional features were needed to produce useful HTML. These features were applied generically as possible and include the introduction of colour. Colour is now supported by the ASCII, PostScript, HTML, and DVI devices. GNU `pic` has also been extended to support colour.

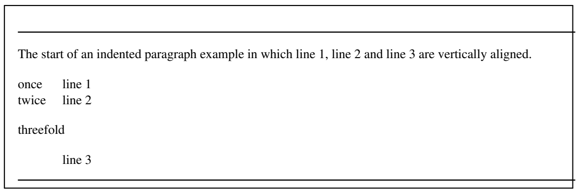
The principles behind the HTML device driver appear sound, however there are still minor bugs in `grohtml`. Future work will include bug fixing and also modifications to `tbl` so that `troff` tables can be translated into HTML `<table>` constructs rather than an image. The remaining macro sets (`mm`, `me`, `mdoc` and `mom`) will also be updated to include the `devtag` requests necessary for `grohtml` to detect section headings, titles and indented paragraphs.

As the OpenOffice file formats are fully published it should be possible to construct a new GNU Troff device driver (`grodoc`) which utilizes the extended `ditroff` intermediate language. This new device driver could be built using a similar design to that of `grohtml`.

ACKNOWLEDGEMENTS

The authors would like to thank all the readers and contributors of the `groff` mailing list for their patience with this project and also for their many valuable bug reports over the last six years. We would also like to thank our respective employers for funding our time while we were working on `groff`.

Finally a great debt of thanks is owed to James Clark without whom there would be no `groff`.



The start of an indented paragraph example in which line 1, line 2 and line 3 are vertically aligned.

```
once    line 1
twice   line 2

threefold

        line 3
```

Figure 12: the result of HTML output from figure 3

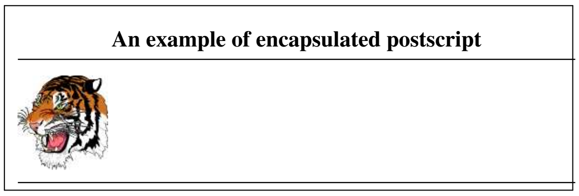
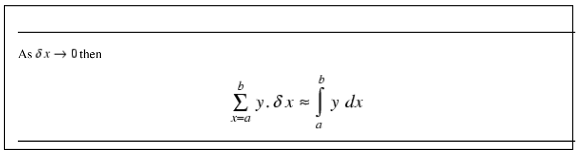


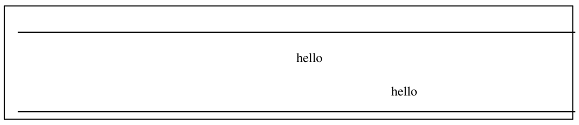
Figure 13: the result of HTML output from figure 5



As $\delta x \to 0$ then

$$\sum_{x=a}^{b} y.\delta x \approx \int_{a}^{b} y\, dx$$

Figure 14: the result of HTML output from figure 7



```
                    hello
                                  hello
```

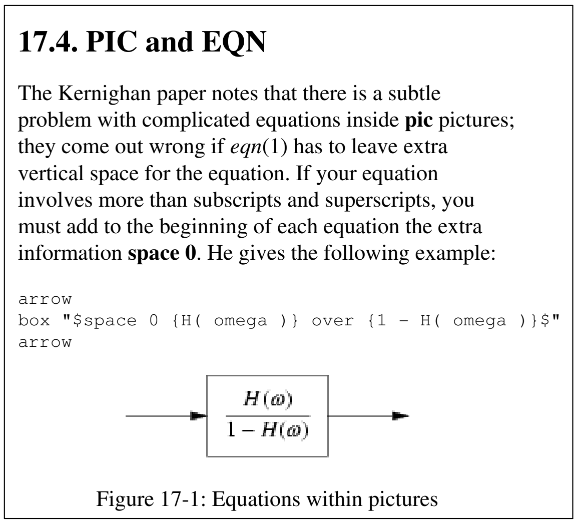
Figure 15: the result of HTML output from figure 9

Tag	Description
`.SH`	start of section heading
`.NH`	start of numbered heading
`.col` *n*	start of column *n*
`.eo.h`	end of heading
`.tl`	start of title
`.eo.tl`	end of title
`.fi` *n*	if *n* is 0 then treat as preformatted text
`.sp`	vertical line space
`.in` *n*	indent next line *n* units
`.ll` *n*	line is *n* units long
`.po` *n*	left offset is *n* units
`.ta` *list*	*list* defines the tab settings
`.ti` *n*	next line is temporarily indented *n* units
`.ce` *n*	centre next *n* lines
`.eol`	end of input line
`.br`	line break forced

Table 2: tags passed from `troff` to `post-grohtml`

`p1`	*page 1*
`V280`	*vertical resolution 280 units/inch*
`H240`	*horizontal resolution 240 units/inch*
`DFd`	*default background colour*
`F ./ex9.n`	*input file ex9.n*
`V360`	*move to vertical position 360 units*
`H720`	*move to horizontal position 3 inches*
`x X devtag:.fi 1`	*word wrap is on*
`x X devtag:.in 720`	*3 inch indentation*
`x X devtag:.ll 1440`	*line length is 6 inches*
`x X devtag:.po 0`	*page offset 0 inches*
`x X devtag:.ce 0`	*centre lines is off*
`x X devtag:.br`	*line break*
`x font 1 R`	*times roman font indexed as the first font*
`f1`	*use times roman font*
`s10`	*point size 10*
`V360`	*move to vertical position 360 units*
`H720`	*move to horizontal position 3 inches*
`md`	*default foreground colour*
`thello`	*emit glyph hello*
`n40 0`	*emit space*
`V400`	*move to vertical position 400 units*
`H960`	*move to horizontal position 4 inches*
`x X devtag:.in 960`	*4 inch indent requested*
`x X devtag:.br`	*line break requested*
`V400`	*move to vertical position 400 units*
`H960`	*move to horizontal position 4 inches*
`thello`	*emit glyph hello*

Figure 16: extended ditroff intermediate output



17.4. PIC and EQN

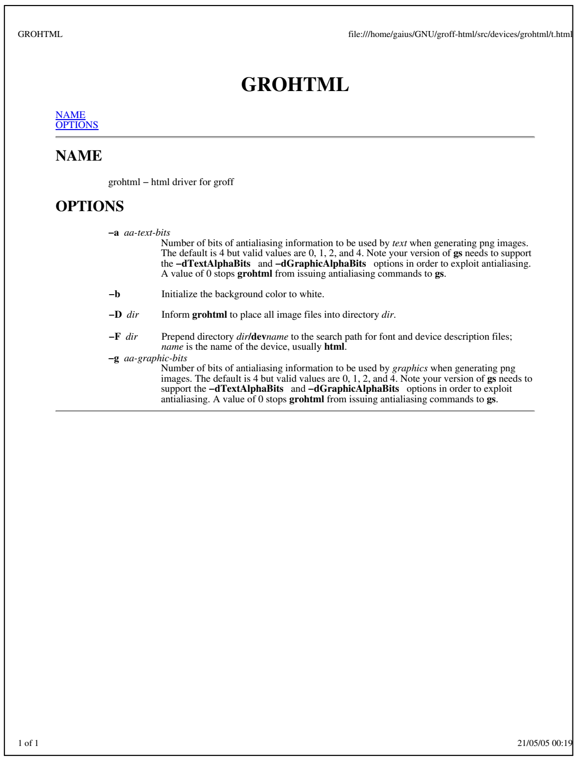
The Kernighan paper notes that there is a subtle problem with complicated equations inside **pic** pictures; they come out wrong if *eqn*(1) has to leave extra vertical space for the equation. If your equation involves more than subscripts and superscripts, you must add to the beginning of each equation the extra information **space 0**. He gives the following example:

```
arrow
box "$space 0 {H( omega )} over {1 - H( omega )}$"
arrow
```

$$\xrightarrow{} \boxed{\frac{H(\omega)}{1 - H(\omega)}} \xrightarrow{}$$

Figure 17-1: Equations within pictures

Figure 17: extract from *"Making Pictures With GNU PIC"* by Eric S. Raymond



GROHTML file:///home/gaius/GNU/groff-html/src/devices/grohtml/t.html

GROHTML

NAME
OPTIONS

NAME

grohtml – html driver for groff

OPTIONS

–a *aa-text-bits*
Number of bits of antialiasing information to be used by *text* when generating png images. The default is 4 but valid values are 0, 1, 2, and 4. Note your version of **gs** needs to support the **–dTextAlphaBits** and **–dGraphicAlphaBits** options in order to exploit antialiasing. A value of 0 stops **grohtml** from issuing antialiasing commands to **gs**.

–b Initialize the background color to white.

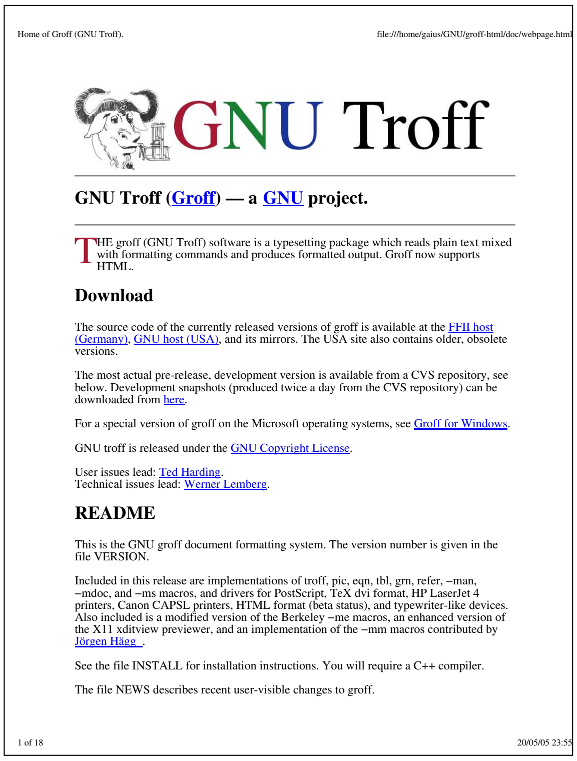
–D *dir* Inform **grohtml** to place all image files into directory *dir*.

–F *dir* Prepend directory *dir***/dev***name* to the search path for font and device description files; *name* is the name of the device, usually **html**.

–g *aa-graphic-bits*
Number of bits of antialiasing information to be used by *graphics* when generating png images. The default is 4 but valid values are 0, 1, 2, and 4. Note your version of **gs** needs to support the **–dTextAlphaBits** and **–dGraphicAlphaBits** options in order to exploit antialiasing. A value of 0 stops **grohtml** from issuing antialiasing commands to **gs**.

1 of 1 21/05/05 00:19

Figure 18: `grohtml` manual page showing vertical alignment of hanging paragraphs



Home of Groff (GNU Troff). file:///home/gaius/GNU/groff-html/doc/webpage.html

GNU Troff

GNU Troff (Groff) — a GNU project.

THE groff (GNU Troff) software is a typesetting package which reads plain text mixed with formatting commands and produces formatted output. Groff now supports HTML.

Download

The source code of the currently released versions of groff is available at the FFII host (Germany), GNU host (USA), and its mirrors. The USA site also contains older, obsolete versions.

The most actual pre-release, development version is available from a CVS repository, see below. Development snapshots (produced twice a day from the CVS repository) can be downloaded from here.

For a special version of groff on the Microsoft operating systems, see Groff for Windows.

GNU troff is released under the GNU Copyright License.

User issues lead: Ted Harding.
Technical issues lead: Werner Lemberg.

README

This is the GNU groff document formatting system. The version number is given in the file VERSION.

Included in this release are implementations of troff, pic, eqn, tbl, grn, refer, –man, –mdoc, and –ms macros, and drivers for PostScript, TeX dvi format, HP LaserJet 4 printers, Canon CAPSL printers, HTML format (beta status), and typewriter-like devices. Also included is a modified version of the Berkeley –me macros, an enhanced version of the X11 xditview previewer, and an implementation of the –mm macros contributed by Jörgen Hägg .

See the file INSTALL for installation instructions. You will require a C++ compiler.

The file NEWS describes recent user-visible changes to groff.

1 of 18 20/05/05 23:55

Figure 19: GNU Troff homepage when printed from a browser

REFERENCES

Allman Eric P. 1980, *Writing Papers with NROFF using −me,* University of California, Berkeley, Berkeley, California 94720.

Kernighan Brian W. 1991, “PIC — A Graphics Language for Typesetting,” Revised User Manual, 116, Bell Labs Computing Science Technical Report.

Kernighan Brian W. 1978, *A TROFF Tutorial,* P. 13, Bell Laboratories, Murray Hill, New Jersey 07974.

Kernighan Brian W. and Cherry Lorinda L. 1977, *A System for Typesetting Mathematics,* Bell Laboratories, Murray Hill, New Jersey 07974.

Kernighan Brian W. and Cherry Lorinda L. 1976, *Typesetting Mathematics User’s Guide,* Bell Laboratories, Murray Hill, New Jersey 07974.

Laumann Oliver 1996, *Unroff - programmable, extensible troff translator.* `http://www-rn.informatik.uni-bremen.de/software/unroff`.

Lesk M. E. 1976, *Tbl — A Program to Format Tables,* Bell Laboratories, Murray Hill, New Jersey 07974.

Lesk M. E. 1976, *Typing Documents on the UNIX System: Using the −ms Macros with Troff and Nroff,* Bell Laboratories, Murray Hill, New Jersey 07974.

Musciano Chuck and Kennedy Bill 1998, *HTML The definitive guide,* Third Edition, O'Reilly & Associates. ISBN 1-68592-492-4.

Ossanna Joseph F. and Kernighan Brian W. 1992, *Nroff/Troff User's Manual,* Bell Laboratories, Murray Hill, New Jersey 07974.

Raymond Eric S. 2005, *Doclifter.* `http://www.catb.org/~esr/doclifter`.

Raymond Eric S. 2005, *Making Pictures With GNU PIC.* `ftp://ftp.gnu.org/gnu/groff/`.

Roelofs Greg 1999, *PNG The Definitive Guide,* First Edition, O'Reilly & Associates. ISBN 1-56592-542-4.

Schaffter Peter 2004, *The Schumann Proof,* Napoleon Publishing. ISBN 1-89491-706-5.

Smith D. W., Mashey J. R., Pariser E. C. and Smith N. W. 1980, *MM — Memorandum Macros,* Bell Laboratories internal memorandum, Murray Hill, New Jersey 07974.

Stevens Richard W. 1998, *UNIX Network Programming,* 1, P. xx, Prentice-Hall International.

Tanenbaum A.S. 1997, *Computer Networks,* 3rd Edition, Prentice-Hall.

Tuthill Bill 1986, *Refer — A Bibliography System,* Computing Services, University of California, Berkeley, CA 94720.

Walsh Norman and Muellner Leonard 1999, *DocBook: The Definitive Guide,* First Edition, O'Reilly & Associates. ISBN: 1-56592-580-7.

Wyk C. J. Van 1982, "A high-level language for specifying pictures," ACM Transactions On Graphics, 1(2), Pp. 163-182.

BIOGRAPHY OF AUTHORS

Gaius Mulley is a senior lecturer at the University of Glamorgan. He is the author of GNU Modula-2 and the groff HTML device driver `grohtml`. His research interests also include performance of microkernels and compiler design. He obtained a BSc(Hons) and PhD in Computer Science from the University of Reading and later worked for Meiko Scientific.

Werner Lemberg (born 1968 in Vienna, Austria) is working as a conductor and singers' coach at the municipal theatre of Koblenz, Germany. In his spare time he maintains the groff package and actively develops the FreeType font rendering library.