NAME

groff_diff − differences between GNU troff and classical troff

DESCRIPTION

This manual page describes the language differences between *groff*, the GNU *roff* text processing system, and the classical *roff* formatter of the freely available Unix 7 of the 1970s, documented in the *Troff User's Manual* by *Ossanna* and *Kernighan*. This includes the roff language as well as the intermediate output format (troff output).

Section "See Also" below gives pointers to both the classical *roff* and the modern *groff* documentation.

GROFF LANGUAGE

In this section, all additional features of *groff* compared to the classical Unix 7 *troff* are described in detail.

Long names

The names of number registers, fonts, strings/macros/diversions, special characters (glyphs), and colors can be of any length. In escape sequences, additionally to the classical '**(***xx*' construction for a two-character glyph name, you can use '**[***xxx***]**' for a name of arbitrary length.

\[*xxx***]** Print the special character (glyph) called *xxx*.

\[*comp1 comp2 . . .***]**
Print composite glyph consisting of multiple components. Example: '\[A ho]' is capital letter A with ogonek which finally maps to glyph name 'u0041_0328'. See*Gr off: The GNU Implementation of troff*, the *groff* Texinfo manual, for details of how a glyph name for a composite glyph is constructed, and **groff_char**(7) for a list of glyph name components used in composite glyph names.

\f[*xxx***]** Set font *xxx*. Additionally, **\f[]** is a new syntax form equal to **\fP**, i.e., to return to the previous font.

[xxx arg1 arg2 . . .***]**
Interpolate string *xxx*, taking *arg1*, *arg2*, . . ., as arguments.

\n[*xxx***]** Interpolate number register *xxx*.

Fractional point sizes

A *scaled point* is equal to **1/sizescale** points, where **sizescale** is specified in the *DESC* file (1 by default). There is a new scale indicator **z** that has the effect of multiplying by sizescale. Requests and escape sequences in troff interpret arguments that represent a point size as being in units of scaled points, but they evaluate each such argument using a default scale indicator of **z**. Arguments treated in this way are the argument to the **ps** request, the third argument to the **cs** request, the second and fourth arguments to the **tkf** request, the argument to the **\H** escape sequence, and those variants of the **\s** escape sequence that take a numeric expression as their argument.

For example, suppose sizescale is 1000; then a scaled point is equivalent to a millipoint; the call **.ps 10.25** is equivalent to **.ps 10.25z** and so sets the point size to 10250 scaled points, which is equal to 10.25 points.

The number register **\n[.s]** returns the point size in points as decimal fraction. There is also a new number register **\n[.ps]** that returns the point size in scaled points.

It would make no sense to use the **z** scale indicator in a numeric expression whose default scale indicator was neither **u** nor **z**, and so **troff** disallows this. Similarly it would make no sense to use a scaling indicator other than **z** or **u** in a numeric expression whose default scale indicator was **z**, and so **troff** disallows this as well.

There is also new scale indicator **s** which multiplies by the number of units in a scaled point. So, for example, **\n[.ps]s** is equal to **1m**. Be sure not to confuse the**s** and **z** scale indicators.

Numeric expressions

Spaces are permitted in a number expression within parentheses.

M indicates a scale of 100ths of an em. **f** indicates a scale of 65536 units, providing fractions for color definitions with the **defcolor** request. For example, 0.5f = 32768u.

e1>?*e2* The maximum of *e1* and *e2*.

e1<?*e2* The minimum of *e1* and *e2*.

(*c***;***e***)** Evaluate *e* using *c* as the default scaling indicator. If*c* is missing, ignore scaling indicators in the evaluation of *e*.

New escape sequences

\A'*anything***'**
: This expands to **1** or **0**, depending on whether *anything* is or is not acceptable as the name of a string, macro, diversion, number register, environment, font, or color. It returns**0** if *anything* is empty. This is useful if you want to look up user input in some sort of associative table.

\B'*anything***'**
: This expands to **1** or **0**, depending on whether *anything* is or is not a valid numeric expression. It returns **0** if *anything* is empty.

\C'*xxx***'**
: Typeset glyph named *xxx*. Normally it is more convenient to use **\[***xxx***]**. But**\C** has the adv antage that it is compatible with recent versions of Unix and is available in compatibility mode.

\E
: This is equivalent to an escape character, but it is not interpreted in copy mode. For example, strings to start and end superscripting could be defined like this

```
.ds { \v'-.3m'\s'\En[.s]*6u/10u'
.ds } \s0\v'.3m'
```

The use of **\E** ensures that these definitions work even if ***{** gets interpreted in copy mode (for example, by being used in a macro argument).

\F *f*
\F(*fm*
\F[*fam***]**
: Change font family. This is the same as the**fam** request. **\F[]** switches back to the pre vious font family (note that **\FP** won't work; it selects font family 'P' instead).

\m*x*
\m(*xx*
\m[*xxx***]**
: Set drawing color. **\m[]** switches back to the previous color.

\M*x*
\M(*xx*
\M[*xxx***]**
: Set background color for filled objects drawn with the **\D'**...**'** commands. **\M[]** switches back to the previous color.

\N'*n***'** Typeset the glyph with index *n* in the current font. *n* can be any integer. Most devices only have glyphs with indices between 0 and 255. If the current font does not contain a glyph with that code, special fonts are *not* searched. The **\N** escape sequence can be conveniently used in conjunction with the **char** request, for example

```
.char \[phone] \f(ZD\N'37'
```

The index of each glyph is given in the fourth column in the font description file after the **charset** command. It is possible to include unnamed glyphs in the font description file by using a name of `---`; the **\N** escape sequence is the only way to use these.

\O*n*
\O[*n***]** Suppress troff output. The escapes **\O2**, **\O3**, **\O4**, and **\O5** are intended for internal use by **grohtml**.

\O0 Disable any ditroff glyphs from being emitted to the device driver, provided that the escape occurs at the outer level (see **\O3** and **\O4**).

\O1 Enable output of glyphs, provided that the escape occurs at the outer level.

\O0 and **\O1** also reset the registers **\n[opminx]**, **\n[opminy]**, **\n[opmaxx]**, and **\n[opmaxy]** to −1. These four registers mark the top left and bottom right hand corners of a box which encompasses all written glyphs.

\O2 Provided that 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. At start-up,**tr off** is at outer level. This is really an internal mechanism for **grohtml** while producing images. They are generated by running the troff source through **troff** to the PostScript device and **ghostscript** to produce images in PNG format. The**\O3** escape starts a ne w page if the device is not html (to reduce the possibility of images crossing a page boundary).

\O4 End a nesting level.

\O5[*P filename***]**
This escape is **grohtml** specific. Provided that this escape occurs at the outer nesting level, write *filename* to stderr . The position of the image,*P*, must be specified and must be one of **l**, **r**, **c**, or **i** (left, right, centered, inline). *filename* is associated with the production of the next inline image.

\R'*name* ±*n***'**
This has the same effect as

.nr *name* ±*n*

\s(*nn*
\s±(*nn* Set the point size to *nn* points; *nn* must be exactly two digits.

\s[±*n***]**
\s±[*n***]**
\s'±*n***'**
\s±'*n***'** Set the point size to *n* scaled points; *n* is a numeric expression with a default scale indicator of **z**.

\V*x*
\V(*xx*
\V[*xxx***]**
Interpolate the contents of the environment variable *xxx*, as returned by **getenv**(3). **\V** is interpreted in copy mode.

\Y*x*
\Y(*xx*
\Y[*xxx***]**
This is approximately equivalent to **\X'*[***xxx***]'**. However the contents of the string or macro *xxx* are not interpreted; also it is permitted for *xxx* to have been defined as a macro and thus contain newlines (it is not permitted for the argument to **\X** to contain newlines). The inclusion of newlines requires an extension to the Unix troff output format, and confuses drivers that do not know about this extension.

\Z'*anything***'**
Print anything and then restore the horizontal and vertical position; *anything* may not contain tabs or leaders.

\$0 The name by which the current macro was invoked. The**als** request can mak e a macro have more than one name.

\$* In a macro or string, the concatenation of all the arguments separated by spaces.

\$@ In a macro or string, the concatenation of all the arguments with each surrounded by double quotes, and separated by spaces.

\$^ In a macro, the representation of all parameters as if they were an argument to the **ds** request.

\$(*nn*

\$[*nnn***]** In a macro or string, this gives the *nn*-th or *nnn*-th argument. Macros and strings can have an unlimited number of arguments.

\?*anything***\?**

When used in a diversion, this transparently embeds *anything* in the diversion. *anything* is read in copy mode. When the diversion is reread, *anything* is interpreted. *anything* may not contain newlines; use **\!** if you want to embed newlines in a diversion. The escape sequence**\?** is also recognized in copy mode and turned into a single internal code; it is this code that terminates *anything*. Thus

```
.nr x 1
.nf
.di d
\?\\?\\\\?\\\\\\\\nx\\\\?\\?\?
.di
.nr x 2
.di e
.d
.di
.nr x 3
.di f
.e
.di
.nr x 4
.f
```

prints **4**.

\/ This increases the width of the preceding glyph so that the spacing between that glyph and the following glyph is correct if the following glyph is a roman glyph. For example, if an italic f is immediately followed by a roman right parenthesis, then in many fonts the top right portion of the f overlaps the top left of the right parenthesis producing *f*), which is ugly. Inserting\/ produces *f*) and avoids this problem. It is a good idea to use this escape sequence whenever an italic glyph is immediately followed by a roman glyph without any intervening space.

\, This modifies the spacing of the following glyph so that the spacing between that glyph and the preceding glyph is correct if the preceding glyph is a roman glyph. For example, inserting **\,** between the parenthesis and the f changes (*f* to (*f*. It is a good idea to use this escape sequence whenever a roman glyph is immediately followed by an italic glyph without any intervening space.

\) Like **\&** except that it behaves like a character declared with the **cflags** request to be transparent for the purposes of end-of-sentence recognition.

\~ This produces an unbreakable space that stretches like a normal inter-word space when a line is adjusted.

\: This causes the insertion of a zero-width break point. It is equal to **\%** within a word but without insertion of a soft hyphen glyph.

\# Everything up to and including the next newline is ignored. This is interpreted in copy mode. It is like \" except that \" does not ignore the terminating newline.

New requests

.aln *xx yy*

Create an alias *xx* for number register object named *yy*. The new name and the old name are exactly equivalent. If*yy* is undefined, a w arning of type **reg** is generated, and the request is ignored.

.als *xx yy*

Create an alias *xx* for request, string, macro, or diversion object named *yy*. The new name and the old name are exactly equivalent (it is similar to a hard rather than a soft link). If *yy* is undefined, a warning of type **mac** is generated, and the request is ignored. The **de**, **am**, **di**, **da**, **ds**, and **as** requests only create a new object if the name of the macro, diversion or string is currently undefined or if it is defined to be a request; normally they modify the value of an existing object.

.am1 *xx yy*

Similar to **.am**, but compatibility mode is switched off during execution. To be more precise, a 'compatibility save' token is inserted at the beginning of the macro addition, and a 'compatibility restore' token at the end. As a consequence, the requests **am**, **am1**, **de**, and **de1** can be intermixed freely since the compatibility save/restore tokens only affect the macro parts defined by **.am1** and **.ds1**.

.ami *xx yy*

Append to macro indirectly. See the**dei** request belo w for more information.

.ami1 *xx yy*

Same as the **ami** request but compatibility mode is switched off during execution.

.as1 *xx yy*

Similar to **.as**, but compatibility mode is switched off during expansion. To be more precise, a 'compatibility save' token is inserted at the beginning of the string, and a 'compatibility restore' token at the end. As a consequence, the requests **as**, **as1**, **ds**, and **ds1** can be intermixed freely since the compatibility save/restore tokens only affect the (sub)strings defined by **as1** and **ds1**.

.asciify *xx*

This request 'unformats' the diversion *xx* in such a way that ASCII and space characters (and some escape sequences) that were formatted and diverted into *xx* are treated like ordinary input characters when *xx* is reread. Useful for diversions in conjunction with the **writem** request. It can be also used for gross hacks; for example, this

```
.tr @.
.di x
@nr n 1
.br
.di
.tr @@
.asciify x
.x
```

sets register **n** to 1. Note that glyph information (font, font size, etc.) is not preserved; use **.unformat** instead.

.backtrace

Print a backtrace of the input stack on stderr.

.blm *xx*

Set the blank line macro to *xx*. If there is a blank line macro, it is invoked when a blank line is encountered instead of the usual troff behaviour.

.box *xx*

.boxa *xx*

These requests are similar to the **di** and **da** requests with the exception that a partially filled line does not become part of the diversion (i.e., the diversion always starts with a new line) but is restored after ending the diversion, discarding the partially filled line which possibly comes from the diversion.

.break Break out of a while loop. See also the **while** and **continue** requests. Be sure not to confuse this with the **br** request.

.brp This is the same as **\p**.

.cflags *n c1 c2 . . .*

Characters *c1*, *c2*, . . ., have properties determined by *n*, which is ORed from the following:

1 The character ends sentences (initially characters **.?!** have this property).

2 Lines can be broken before the character (initially no characters have this property); a line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This can be overridden with value 64.

4 Lines can be broken after the character (initially characters **−\[hy]\[em]** have this property); a line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This can be overridden with value 64.

8 The glyph associated with this character overlaps horizontally (initially characters **\[ul]\[rn]\[ru]\[radicalex]\[sqrtex]** have this property).

16 The glyph associated with this character overlaps vertically (initially glyph **\[br]** has this property).

32 An end-of-sentence character followed by any number of characters with this property is treated as the end of a sentence if followed by a newline or two spaces; in other words the character is transparent for the purposes of end-of-sentence recognition; this is the same as having a zero space factor in TEX (initially characters **"')]*\[dg]\[rq]\[cq]** have this property).

64 Ignore hyphenation code values of the surrounding characters. Use this in combination with values 2 and 4 (initially no characters have this property).

128 Prohibit a line break before the character, but allow a line break after the character. This works only in combination with flags 256 and 512 and has no effect otherwise.

256 Prohibit a line break after the character, but allow a line break before the character. This works only in combination with flags 128 and 512 and has no effect otherwise.

512 Allow line break before or after the character. This works only in combination with flags 128 and 256 and has no effect otherwise.

Contrary to flag values 2 and 4, the flags 128, 256, and 512 work pairwise. If, for example, the left character has value 512, and the right character 128, no line break gets inserted. If we use value 6 instead for the left character, a line break after the character can't be suppressed since the right neighbour character doesn't get examined.

.char *c string*

[This request can both define characters and glyphs.]

Define entity *c* to be *string*. To be more precise, define (or even override) a groff entity which can be accessed with name *c* on the input side, and which uses *string* on the output side. Every time glyph *c* needs to be printed, *string* is processed in a temporary environment and the result is wrapped up into a single object. Compatibility mode is turned off and the escape character is set to \ while *string* is being processed. Any emboldening, constant spacing or track kerning is applied to this object rather than to individual glyphs in *string*.

A groff object defined by this request can be used just like a normal glyph provided by the output device. In particular other characters can be translated to it with the **tr** request; it can be made the leader glyph by the **lc** request; repeated patterns can be drawn with the glyph using the **\l** and **\L** escape sequences; words containing *c* can be hyphenated correctly, if the **hcode** request is used to give the object a hyphenation code.

There is a special anti-recursion feature: Use of glyph within the glyph's definition is handled like normal glyphs not defined with **char**.

A glyph definition can be removed with the **rchar** request.

.chop *xx*

Chop the last element off macro, string, or diversion *xx*. This is useful for removing the newline from the end of diversions that are to be interpolated as strings.

.class *name c1 c2 . . .*

Assign *name* to a set of characters *c1*, *c2*, . . ., so that they can be referred to from other requests easily (currently **.cflags** only). Character ranges (indicated by an intermediate '−') and nested classes are possible also. This is useful to assign properties to a large set of characters.

.close *stream*

Close the stream named *stream*; *stream* will no longer be an acceptable argument to the **write** request. See the**open** request.

.composite *glyph1 glyph2*

Map glyph name *glyph1* to glyph name *glyph2* if it is used in **\[. . .]** with more than one component.

.continue

Finish the current iteration of a while loop. See also the **while** and **break** requests.

.color *n*

If *n* is non-zero or missing, enable colors (this is the default), otherwise disable them.

.cp *n* If *n* is non-zero or missing, enable compatibility mode, otherwise disable it. In compatibility mode, long names are not recognized, and the incompatibilities caused by long names do not arise.

.defcolor *xxx scheme color_components*

Define color *xxx*. *scheme* can be one of the following values: **rgb** (three components), **cmy** (three components), **cmyk** (four components), and **gray** or **grey** (one component). Color components can be given either as a hexadecimal string or as positive decimal integers in the range 0–65535. A hexadecimal string contains all color components concatenated; it must start with either **#** or **##**. The former specifies hex values in the range 0–255 (which are internally multiplied by 257), the latter in the range 0–65535. Examples: #FFC0CB (pink), ##ffff0000ffff (magenta). A new scaling indicator **f** has been introduced which multiplies its value by 65536; this makes it convenient to specify color components as fractions in the range 0 to 1. Example:

```
.defcolor darkgreen rgb 0.1f 0.5f 0.2f
```

Note that **f** is the default scaling indicator for the **defcolor** request, thus the above statement is equivalent to

```
.defcolor darkgreen rgb 0.1 0.5 0.2
```

The color named **default** (which is device-specific) can't be redefined. It is possible that the default color for **\M** and **\m** is not the same.

.de1 *xx yy*

Similar to **.de**, but compatibility mode is switched off during execution. On entry, the current compatibility mode is saved and restored at exit.

.dei *xx yy*

Define macro indirectly. The following example

```
.ds xx aa
.ds yy bb
.dei xx yy
```

is equivalent to

```
.de aa bb
```

.dei1 *xx yy*

Similar to the **dei** request but compatibility mode is switched off during execution.

.device *anything*
This is (almost) the same as the **X** escape. *anything* is read in copy mode; a leading **"** is stripped.

.devicem *xx*
This is the same as the **Y** escape (to embed the contents of a macro into the intermediate output preceded with 'x X').

.do *xxx* Interpret *.xxx* with compatibility mode disabled. For example,

```
.do fam T
```

would have the same effect as

```
.fam T
```

except that it would work even if compatibility mode had been enabled. Note that the previous compatibility mode is restored before any files sourced by *xxx* are interpreted.

.ds1 *xx yy*
Similar to **.ds**, but compatibility mode is switched off during expansion. To be more precise, a 'compatibility save' token is inserted at the beginning of the string, and a 'compatibility restore' token at the end.

.ecs Save current escape character.

.ecr Restore escape character saved with **ecs**. Without a previous call to **ecs**, '\' will be the new escape character.

.evc *xx* Copy the contents of environment *xx* to the current environment. No pushing or popping of environments is done.

.fam *xx*
Set the current font family to *xx*. The current font family is part of the current environment. If*xx* is missing, switch back to previous font family. The value at start-up is 'T'. See the description of the **sty** request for more information on font families.

.fchar *c string*
Define fallback character (or glyph) *c* to be *string*. The syntax of this request is the same as the **char** request; the only difference is that a glyph defined with **char** hides the glyph with the same name in the current font, whereas a glyph defined with **fchar** is checked only if the particular glyph isn't found in the current font. This test happens before checking special fonts.

.fcolor *c*
Set the fill color to *c*. If*c* is missing, switch to the pre vious fill color.

.fschar *f c string*
Define fallback character (or glyph) *c* for font *f* to be *string*. The syntax of this request is the same as the **char** request (with an additional argument to specify the font); a glyph defined with **fschar** is searched after the list of fonts declared with the **fspecial** request but before the list of fonts declared with **.special**.

.fspecial *f s1 s2 ...*
When the current font is *f*, fonts *s1*, *s2*, ..., are special, that is, the y are searched for glyphs not in the current font. Any fonts specified in the **special** request are searched after fonts specified in the **fspecial** request. Without argument, reset the list of global special fonts to be empty.

.ftr *f g* Translate font *f* to *g*. Whene ver a font named *f* is referred to in an **f** escape sequence, in the **F** and **S** conditional operators, or in the **ft**, **ul**, **bd**, **cs**, **tkf**, **special**, **fspecial**, **fp**, or **sty** requests, font *g* is used. If *g* is missing, or equal to *f* then font *f* is not translated.

.fzoom *f zoom*
Set zoom factor *zoom* for font *f*. *zoom* must a non-negative integer multiple of 1/1000th. If it is missing or is equal to zero, it means the same as 1000, namely no magnification. *f* must be a real font name, not a style.

.gcolor *c*

Set the glyph color to *c*. If *c* is missing, switch to the pre vious glyph color.

.hcode *c1 code1 c2 code2* ...

Set the hyphenation code of character *c1* to *code1* and that of *c2* to *code2*, and so on. A hyphenation code must be a single input character (not a special character) other than a digit or a space. Initially each lower-case letter a–z has a hyphenation code, which is itself, and each upper-case letter A–Z has a hyphenation code which is the lower-case version of itself. See also the **hpf** request.

.hla *lang*

Set the current hyphenation language to *lang*. Hyphenation exceptions specified with the **hw** request and hyphenation patterns specified with the **hpf** request are both associated with the current hyphenation language. The **hla** request is usually invoked by the **troffrc** file to set up a default language.

.hlm *n* Set the maximum number of consecutive hyphenated lines to *n*. If *n* is ne gative, there is no maximum. The default value is −1. This value is associated with the current environment. Only lines output from an environment count towards the maximum associated with that environment. Hyphens resulting from **\%** are counted; explicit hyphens are not.

.hpf *file*

Read hyphenation patterns from *file*; this is searched for in the same w ay that *name***.tmac** is searched for when the **−m***name* option is specified. It should have the same format as (simple) TEX patterns files. More specifically, the following scanning rules are implemented.

- A percent sign starts a comment (up to the end of the line) even if preceded by a backslash.
- No support for 'digraphs' like **\$**.
- ^^*xx* (*x* is 0–9 or a–f) and ^^*x* (character code of *x* in the range 0–127) are recognized; other use of ^ causes an error.
- No macro expansion.
- **hpf** checks for the expression **\patterns{...}** (possibly with whitespace before and after the braces). Everything between the braces is taken as hyphenation patterns. Consequently, **{** and **}** are not allowed in patterns.
- Similarly, **\hyphenation{...}** gives a list of hyphenation exceptions.
- **\endinput** is recognized also.
- For backwards compatibility, if **\patterns** is missing, the whole file is treated as a list of hyphenation patterns (only recognizing the **%** character as the start of a comment).

Use the **hpfcode** request to map the encoding used in hyphenation patterns files to **groff**'s input encoding. By default, everything maps to itself except letters 'A' to 'Z' which map to 'a' to 'z'.

The set of hyphenation patterns is associated with the current language set by the **hla** request. The **hpf** request is usually invoked by the **troffrc** file; a second call replaces the old patterns with the new ones.

.hpfa *file*

The same as **hpf** except that the hyphenation patterns from *file* are appended to the patterns already loaded in the current language.

.hpfcode *a b c d* ...

After reading a hyphenation patterns file with the **hpf** or **hpfa** request, convert all characters with character code *a* in the recently read patterns to character code *b*, character code *c* to *d*, etc. Initially, all character codes map to themselves. The arguments of **hpfcode** must be integers in the range 0 to 255. Note that it is even possible to use character codes which are invalid in **groff** otherwise.

.hym *n* Set the *hyphenation margin* to *n*: when the current adjustment mode is not **b**, the line is not hyphenated if the line is no more than *n* short. The default hyphenation margin is 0. The default scaling indicator for this request is **m**. The hyphenation margin is associated with the current environment. The current hyphenation margin is available in the **\n[.hym]** register.

.hys *n* Set the *hyphenation space* to *n*: When the current adjustment mode is **b** don't hyphenate the line if the line can be justified by adding no more than *n* extra space to each word space. The default hyphenation space is 0. The default scaling indicator for this request is **m**. The hyphenation space is associated with the current environment. The current hyphenation space is available in the **\n[.hys]** register.

.itc *n macro*
Variant of **.it** for which a line interrupted with **\c** is not counted as an input line.

.kern *n* If *n* is non-zero or missing, enable pairwise kerning, otherwise disable it.

.length *xx string*
Compute the length of *string* and return it in the number register *xx* (which is not necessarily defined before).

.linetabs *n*
If *n* is non-zero or missing, enable line-tabs mode, otherwise disable it (which is the default). In line-tabs mode, tab distances are computed relative to the (current) output line. Otherwise they are taken relative to the input line. For example, the following

```
.ds x a\t\c
.ds y b\t\c
.ds z c
.ta 1i 3i
\*x
\*y
\*z
```

yields

```
a          b          c
```

In line-tabs mode, the same code gives

```
a          b                     c
```

Line-tabs mode is associated with the current environment; the read-only number register **\n[.linetabs]** is set to 1 if in line-tabs mode, and 0 otherwise.

.lsm *xx* Set the leading spaces macro to *xx*. If there are leading spaces in an input line, it is invoked instead of the usual troff behaviour; the leading spaces are removed. Registers **\n[lsn]** and **\n[lss]** hold the number of removed leading spaces and the corresponding horizontal space, respectively.

.mso *file*
The same as the **so** request except that *file* is searched for in the same directories as macro files for the **−m** command-line option. If the file name to be included has the form *name***.tmac** and it isn't found, **mso** tries to include **tmac.***name* instead and vice versa. A warning of type **file** is generated if *file* can't be loaded, and the request is ignored.

.nop *anything*
Execute *anything*. This is similar to '.if 1'.

.nroff Make the **n** built-in condition true and the **t** built-in condition false. This can be reversed using the **troff** request.

.open *stream filename*
Open *filename* for writing and associate the stream named *stream* with it. See also the **close** and **write** requests.

.opena *stream filename*
Like **open**, but if *filename* e xists, append to it instead of truncating it.

.output *string*
Emit *string* directly to the intermediate output (subject to copy-mode interpretation); this is similar to **\!** used at the top level. An initial double quote in*string* is stripped of f to allow initial blanks.

.pev Print the current environment and each defined environment state on stderr.

.pnr Print the names and contents of all currently defined number registers on stderr.

.psbb *filename*
Get the bounding box of a PostScript image *filename*. This file must conform to Adobe's Document Structuring Conventions; the command looks for a **%%BoundingBox** comment to extract the bounding box values. After a successful call, the coordinates (in PostScript units) of the lower left and upper right corner can be found in the registers **\n[llx]**, **\n[lly]**, **\n[urx]**, and **\n[ury]**, respectively. If some error has occurred, the four registers are set to zero.

.pso *command*
This behaves like the **so** request except that input comes from the standard output of *command*.

.ptr Print the names and positions of all traps (not including input line traps and diversion traps) on stderr. Empty slots in the page trap list are printed as well, because they can affect the priority of subsequently planted traps.

.pvs ±*n* Set the post-vertical line space to *n*; default scale indicator is **p**. This value is added to each line after it has been output. With no argument, the post-vertical line space is set to its previous value.

The total vertical line spacing consists of four components: **.vs** and **\x** with a negative value which are applied before the line is output, and **.pvs** and **\x** with a positive value which are applied after the line is output.

.rchar *c1 c2* . . .
Remove the definitions of glyphs *c1*, *c2*, . . . This undoes the effect of a **char** request.

.return Within a macro, return immediately. If called with an argument, return twice, namely from the current macro and from the macro one level higher. No effect otherwise.

.rfschar *c1 c2* . . .
Remove the font-specific definitions of glyphs *c1*, *c2*, . . . This undoes the effect of an **fschar** request.

.rj
.rj *n* Right justify the next *n* input lines. Without an argument right justify the next input line. The number of lines to be right justified is available in the **\n[.rj]** register. This implicitly does**.ce 0**. The **ce** request implicitly does **.rj 0**.

.rnn *xx yy*
Rename number register *xx* to *yy*.

.schar *c string*
Define global fallback character (or glyph) *c* to be *string*. The syntax of this request is the same as the **char** request; a glyph defined with **schar** is searched after the list of fonts declared with the **special** request but before the mounted special fonts.

.shc *c* Set the soft hyphen character to *c*. If*c* is omitted, the soft h yphen character is set to the default **\[hy]**. The soft hyphen character is the glyph which is inserted when a word is hyphenated at a line break. If the soft hyphen character does not exist in the font of the glyph immediately preceding a potential break point, then the line is not broken at that point. Neither definitions (specified with the **char** request) nor translations (specified with the **tr** request) are considered when finding the soft hyphen character.

.shift *n* In a macro, shift the arguments by *n* positions: argument *i* becomes argument $i - n$; arguments 1 to *n* are no longer available. If*n* is missing, ar guments are shifted by 1. Shifting by negative

amounts is currently undefined.

.sizes *s1 s2 . . . sn* **[0]**
This command is similar to the **sizes** command of a *DESC* file. It sets the available font sizes for the current font to *s1*, *s2*, . . . , *sn* scaled points. The list of sizes can be terminated by an optional **0**. Each*si* can also be a range of sizes *m–n*. Contrary to the font file command, the list can't extend over more than a single line.

.special *s1 s2 . . .*
Fonts *s1*, *s2*, . . ., are special and are searched for glyphs not in the current font. Without arguments, reset the list of special fonts to be empty.

.spreadwarn *limit*
Make **troff** emit a warning if the additional space inserted for each space between words in an output line is larger or equal to *limit*. A negative value is changed to zero; no argument toggles the warning on and off without changing *limit*. The default scaling indicator is **m**. At startup, **spreadwarn** is deactivated, and *limit* is set to 3m. For example, **.spreadwarn 0.2m** causes a warning if **troff** must add 0.2m or more for each interword space in a line. This request is active only if text is justified to both margins (using **.ad b**).

.sty *n f* Associate style *f* with font position *n*. A font position can be associated either with a font or with a style. The current font is the index of a font position and so is also either a font or a style. When it is a style, the font that is actually used is the font the name of which is the concatenation of the name of the current family and the name of the current style. For example, if the current font is 1 and font position 1 is associated with style **R** and the current font family is **T**, then font **TR** is used. If the current font is not a style, then the current family is ignored. When the requests **cs**, **bd**, **tkf**, **uf**, or **fspecial** are applied to a style, then they are applied instead to the member of the current family corresponding to that style. The default family can be set with the **−f** command-line option. The **styles** command in the DESC file controls which font positions (if any) are initially associated with styles rather than fonts.

.substring *xx n1* **[***n2***]**
Replace the string named *xx* with the substring defined by the indices *n1* and *n2*. The first character in the string has index 0. If*n2* is omitted, it is tak en to be equal to the string's length. If the index value *n1* or *n2* is negative, it is counted from the end of the string, going backwards: The last character has index −1, the character before the last character has index −2, etc.

.tkf *f s1 n1 s2 n2*
Enable track kerning for font *f*. When the current font is*f* the width of e very glyph is increased by an amount between *n1* and *n2*; when the current point size is less than or equal to *s1* the width is increased by *n1*; when it is greater than or equal to *s2* the width is increased by *n2*; when the point size is greater than or equal to *s1* and less than or equal to *s2* the increase in width is a linear function of the point size.

.tm1 *string*
Similar to the **tm** request, *string* is read in copy mode and written on the standard error, but an initial double quote in *string* is stripped off to allow initial blanks.

.tmc *string*
Similar to **tm1** but without writing a final newline.

.trf *filename*
Transparently output the contents of file *filename*. Each line is output as if preceded by**\!**; ho wever, the lines are not subject to copy-mode interpretation. If the file does not end with a newline, then a newline is added. For example, you can define a macro *x* containing the contents of file *f*, using

```
.di x
.trf f
.di
```

Unlike with the **cf** request, the file cannot contain characters, such as NUL, that are not valid troff input characters.

.trin *abcd*

This is the same as the **tr** request except that the **asciify** request uses the character code (if any) before the character translation. Example:

```
.trin ax
.di xxx
a
.br
.di
.xxx
.trin aa
.asciify xxx
.xxx
```

The result is **x a**. Using**tr**, the result w ould be **x x**.

.trnt *abcd*

This is the same as the **tr** request except that the translations do not apply to text that is transparently throughput into a diversion with **\!**. For example,

```
.tr ab
.di x
\!.tm a
.di
.x
```

prints **b**; if **trnt** is used instead of **tr** it prints **a**.

.troff Make the **n** built-in condition false, and the **t** built-in condition true. This undoes the effect of the **nroff** request.

.unformat *xx*

This request 'unformats' the diversion *xx*. Contrary to the**asciify** request, which tries to con vert formatted elements of the diversion back to input tokens as much as possible, **.unformat** only handles tabs and spaces between words (usually caused by spaces or newlines in the input) specially. The former are treated as if they were input tokens, and the latter are stretchable again. Note that the vertical size of lines is not preserved. Glyph information (font, font size, space width, etc.) is retained. Useful in conjunction with the**box** and **boxa** requests.

.vpt *n* Enable vertical position traps if *n* is non-zero, disable them otherwise. Vertical position traps are traps set by the **wh** or **dt** requests. Traps set by the **it** request are not vertical position traps. The parameter that controls whether vertical position traps are enabled is global. Initially vertical position traps are enabled.

.warn *n*

Control warnings. *n* is the sum of the numbers associated with each warning that is to be enabled; all other warnings are disabled. The number associated with each warning is listed in **troff**(1). For example, **.warn 0** disables all warnings, and **.warn 1** disables all warnings except that about missing glyphs. If *n* is not given, all warnings are enabled.

.warnscale *si*

Set the scaling indicator used in warnings to *si*. Valid values for *si* are **u**, **i**, **c**, **p**, and **P**. At startup, it is set to **i**.

.while *c anything*

While condition *c* is true, accept *anything* as input; *c* can be any condition acceptable to an **if** request; *anything* can comprise multiple lines if the first line starts with **\{** and the last line ends with **\}**. See also the**br eak** and **continue** requests.

.write *stream anything*
: Write *anything* to the stream named *stream*. *stream* must previously have been the subject of an **open** request. *anything* is read in copy mode; a leading **"** is stripped.

.writec *stream anything*
: Similar to **write** but without writing a final newline.

.writem *stream xx*
: Write the contents of the macro or string *xx* to the stream named *stream*. *stream* must previously have been the subject of an **open** request. *xx* is read in copy mode.

Extended escape sequences

\D'...**'** All drawing commands of groff's intermediate output are accepted. See subsection "Drawing Commands" below.

Extended requests

.cf *filename*
: When used in a diversion, this embeds in the diversion an object which, when reread, will cause the contents of *filename* to be transparently copied through to the output. In Unix troff, the contents of *filename* is immediately copied through to the output regardless of whether there is a current diversion; this behaviour is so anomalous that it must be considered a bug.

.de *xx yy*
.am *xx yy*
.ds *xx yy*
.as *xx yy*
: In compatibility mode, these requests behaves similar to **.de1**, **.am1**, **.ds1**, and **.as1**, respectively: A 'compatibility save' token is inserted at the beginning, and a 'compatibility restore' token at the end, with compatibility mode switched on during execution.

.ev *xx* If *xx* is not a number, this switches to a named environment called *xx*. The environment should be popped with a matching **ev** request without any arguments, just as for numbered environments. There is no limit on the number of named environments; they are created the first time that they are referenced.

.hy *n* New additive values 16 and 32 are available; the former enables hyphenation before the last character, the latter enables hyphenation after the first character.

.ss *m n* When two arguments are given to the **ss** request, the second argument gives the *sentence space size*. If the second argument is not given, the sentence space size is the same as the word space size. Like the word space size, the sentence space is in units of one twelfth of the spacewidth parameter for the current font. Initially both the word space size and the sentence space size are 12. Contrary to Unix troff, GNU troff handles this request in nroff mode also; a given value is then rounded down to the nearest multiple of 12. The sentence space size is used in two circumstances. If the end of a sentence occurs at the end of a line in fill mode, then both an inter-word space and a sentence space are added; if two spaces follow the end of a sentence in the middle of a line, then the second space is a sentence space. Note that the behaviour of Unix troff is exactly that exhibited by GNU troff if a second argument is never given to the **ss** request. In GNU troff, as in Unix troff, you should always follow a sentence with either a newline or two spaces.

.ta *n1 n2* ... *nn* **T** *r1 r2* ... *rn*
: Set tabs at positions *n1*, *n2*, ..., *nn* and then set tabs at *nn* + *r1*, *nn* + *r2*, ..., *nn* + *rn* and then at *nn* + *rn* + *r1*, *nn* + *rn* + *r2*, ..., *nn* + *rn* + *rn*, and so on. For example,

```
.ta T .5i
```

sets tabs every half an inch.

New number registers

The following read-only registers are available:

\n[.br] Within a macro call, it is set to 1 if the macro is called with the 'normal' control character ('.' by default), and set to 0 otherwise. This allows the reliable modification of requests.

```
.als bp*orig bp
.de bp
.tm before bp
.ie \\n[.br] .bp*orig
.el 'bp*orig
.tm after bp
..
```

Using this register outside of a macro makes no sense (it always returns zero in such cases).

\n[.C] 1 if compatibility mode is in effect, 0 otherwise.

\n[.cdp]
The depth of the last glyph added to the current environment. It is positive if the glyph extends below the baseline.

\n[.ce] The number of lines remaining to be centered, as set by the **ce** request.

\n[.cht] The height of the last glyph added to the current environment. It is positive if the glyph extends above the baseline.

\n[.color]
1 if colors are enabled, 0 otherwise.

\n[.csk]
The skew of the last glyph added to the current environment. Thes*k ew* of a glyph is how far to the right of the center of a glyph the center of an accent over that glyph should be placed.

\n[.ev] The name or number of the current environment. This is a string-valued register.

\n[.fam]
The current font family. This is a string-valued register.

\n[.fn] The current (internal) real font name. This is a string-valued register. If the current font is a style, the value of **\n[.fn]** is the proper concatenation of family and style name.

\n[.fp] The number of the next free font position.

\n[.g] Always 1. Macros should use this to determine whether they are running under GNU troff.

\n[.height]
The current height of the font as set with **\H**.

\n[.hla] The current hyphenation language as set by the **hla** request.

\n[.hlc] The number of immediately preceding consecutive hyphenated lines.

\n[.hlm]
The maximum allowed number of consecutive hyphenated lines, as set by the **hlm** request.

\n[.hy] The current hyphenation flags (as set by the **hy** request).

\n[.hym]
The current hyphenation margin (as set by the **hym** request).

\n[.hys]
The current hyphenation space (as set by the **hys** request).

\n[.in] The indentation that applies to the current output line.

\n[.int] Set to a positive value if last output line is interrupted (i.e., if it contains **\c**).

\n[.kern]
1 if pairwise kerning is enabled, 0 otherwise.

\n[.lg] The current ligature mode (as set by the **lg** request).

\n[.linetabs]
The current line-tabs mode (as set by the **linetabs** request).

\n[.ll] The line length that applies to the current output line.

\n[.lt] The title length as set by the **lt** request.

\n[.m] The name of the current drawing color. This is a string-valued register.

\n[.M] The name of the current background color. This is a string-valued register.

\n[.ne] The amount of space that was needed in the last **ne** request that caused a trap to be sprung. Useful in conjunction with the **\n[.trunc]** register.

\n[.ns] 1 if no-space mode is active, 0 otherwise.

\n[.O] The current output level as set with **\O**.

\n[.P] 1 if the current page is in the output list set with **−o**.

\n[.pe] 1 during a page ejection caused by the **bp** request, 0 otherwise.

\n[.pn] The number of the next page, either the value set by a **pn** request, or the number of the current page plus 1.

\n[.ps] The current point size in scaled points.

\n[.psr]
The last-requested point size in scaled points.

\n[.pvs]
The current post-vertical line space as set with the **pvs** request.

\n[.rj] The number of lines to be right-justified as set by the **rj** request.

\n[.slant]
The slant of the current font as set with **\S**.

\n[.sr] The last requested point size in points as a decimal fraction. This is a string-valued register.

\n[.ss]
\n[.sss] These give the values of the parameters set by the first and second arguments of the **ss** request.

\n[.sty] The current font style. This is a string-valued register.

\n[.tabs]
A string representation of the current tab settings suitable for use as an argument to the **ta** request.

\n[.trunc]
The amount of vertical space truncated by the most recently sprung vertical position trap, or, if the trap was sprung by an **ne** request, minus the amount of vertical motion produced by the **ne** request. In other words, at the point a trap is sprung, it represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. Useful in conjunction with the **\n[.ne]** register.

\n[.U] Set to 1 if in safer mode and to 0 if in unsafe mode (as given with the **−U** command-line option).

\n[.vpt]
1 if vertical position traps are enabled, 0 otherwise.

\n[.warn]
The sum of the numbers associated with each of the currently enabled warnings. The number associated with each warning is listed in **troff**(1).

\n[.x] The major version number. For example, if the version number is 1.03, then **\n[.x]** contains 1.

\n[.y] The minor version number. For example, if the version number is 1.03, then **\n[.y]** contains 03.

\n[.Y] The revision number of groff.

\n[.zoom]
The zoom value of the current font, in multiples of 1/1000th. Zero if no magnification.

\n[llx]
\n[lly]
\n[urx]
\n[ury] These four read/write registers are set by the **psbb** request and contain the bounding box values (in PostScript units) of a given PostScript image.

The following read/write registers are set by the **\w** escape sequence:

\n[rst]
\n[rsb] Like the **st** and **sb** registers, but take account of the heights and depths of glyphs.

\n[ssc] The amount of horizontal space (possibly negative) that should be added to the last glyph before a subscript.

\n[skw]
How far to right of the center of the last glyph in the **\w** argument, the center of an accent from a roman font should be placed over that glyph.

Other available read/write number registers are:

\n[c.] The current input line number. **\n[.c]** is a read-only alias to this register.

\n[hours]
The number of hours past midnight. Initialized at start-up.

\n[hp] The current horizontal position at input line.

\n[lsn]
\n[lss] If there are leading spaces in an input line, these registers hold the number of leading spaces and the corresponding horizontal space, respectively.

\n[minutes]
The number of minutes after the hour. Initialized at start-up.

\n[seconds]
The number of seconds after the minute. Initialized at start-up.

\n[systat]
The return value of the system() function executed by the last **sy** request.

\n[slimit]
If greater than 0, the maximum number of objects on the input stack. If less than or equal to 0, there is no limit on the number of objects on the input stack. With no limit, recursion can continue until virtual memory is exhausted.

\n[year]
The current year. Note that the traditional**tr off** number register **\n[yr]** is the current year minus 1900.

Miscellaneous

troff predefines a single (read/write) string-based register, ***[.T]**, which contains the argument given to the **−T** command-line option, namely the current output device (for example, *latin1* or *ascii*). Note that this is not the same as the (read-only) number register **\n[.T]** which is defined to be 1 if **troff** is called with the **−T** command-line option, and zero otherwise. This behaviour is different from Unix troff.

Fonts not listed in the DESC file are automatically mounted on the next available font position when they are referenced. If a font is to be mounted explicitly with the **fp** request on an unused font position, it should be mounted on the first unused font position, which can be found in the **\n[.fp]** register; although **troff** does not enforce this strictly, it does not allow a font to be mounted at a position whose number is much greater than that of any currently used position.

Interpolating a string does not hide existing macro arguments. Thus in a macro, a more efficient way of doing

.*xx* **$@**

is

[xx***]**\\

If the font description file contains pairwise kerning information, glyphs from that font are kerned. Kerning between two glyphs can be inhibited by placing a **\&** between them.

In a string comparison in a condition, characters that appear at different input levels to the first delimiter character are not recognized as the second or third delimiters. This applies also to the **tl** request. In a **\w** escape sequence, a character that appears at a different input level to the starting delimiter character is not recognized as the closing delimiter character. The same is true for**\A**, **\b**, **\B**, **\C**, **\l**, **\L**, **\o**, **\X**, and **\Z**. When decoding a macro or string argument that is delimited by double quotes, a character that appears at a different input level to the starting delimiter character is not recognized as the closing delimiter character. The implementation of **\$@** ensures that the double quotes surrounding an argument appear at the same input level, which is different to the input level of the argument itself. In a long escape name **]** is not recognized as a closing delimiter except when it occurs at the same input level as the opening **[**. In compatibility mode, no attention is paid to the input-level.

There are some new types of condition:

.if r*xxx* True if there is a number register named *xxx*.

.if d*xxx* True if there is a string, macro, diversion, or request named *xxx*.

.if m*xxx*
True if there is a color named *xxx*.

.if c*ch* True if there is a character (or glyph) *ch* available; *ch* is either an ASCII character or a glyph (special character) **\N'***xxx***'**, **\(***xx* or **\[***xxx***]**; the condition is also true if *ch* has been defined by the **char** request.

.if F *f* True if font *f* e xists. **f** is handled as if it was opened with the **ft** request (this is, font translation and styles are applied), without actually mounting it.

.if S*s* True if style *s* has been registered. Font translation is applied.

The **tr** request can now map characters onto **\~**.

The space width emitted by the \| and \^ escape sequences can be controlled on a per-font basis. If there is a glyph named \| or \^, respectively (note the leading backslash), defined in the current font file, use this glyph's width instead of the default value.

It is now possible to have whitespace between the first and second dot (or the name of the ending macro) to end a macro definition. Example:

```
.if t \{\
.  de bar
.    nop Hello, I'm 'bar'.
.  .
.\}
```

INTERMEDIATE OUTPUT FORMAT

This section describes the format output by GNU troff. The output format used by GNU troff is very similar to that used by Unix device-independent troff. Only the differences are documented here.

Units

The argument to the **s** command is in scaled points (units of points/*n*, where *n* is the argument to the **sizescale** command in the DESC file). The argument to the **x Height** command is also in scaled points.

Text Commands

N*n* Print glyph with index *n* (a non-negative integer) of the current font.

If the **tcommand** line is present in the DESC file, troff uses the following two commands.

t*xxx* *xxx* is any sequence of characters terminated by a space or a newline (to be more precise, it is a sequence of glyphs which are accessed with the corresponding characters); the first character should be printed at the current position, the current horizontal position should be increased by the width of the first character, and so on for each character. The width of the glyph is that given in the font file, appropriately scaled for the current point size, and rounded so that it is a multiple of the horizontal resolution. Special characters cannot be printed using this command.

u*n xxx* This is same as the **t** command except that after printing each character, the current horizontal position is increased by the sum of the width of that character and *n*.

Note that single characters can have the eighth bit set, as can the names of fonts and special characters.

The names of glyphs and fonts can be of arbitrary length; drivers should not assume that they are only two characters long.

When a glyph is to be printed, that glyph is always in the current font. Unlike device-independent troff, it is not necessary for drivers to search special fonts to find a glyph.

For color support, some new commands have been added:

mc *cyan magenta yellow*
md
mg *gray*
mk *cyan magenta yellow black*
mr *red green blue*
Set the color components of the current drawing color, using various color schemes. **md** resets the drawing color to the default value. The arguments are integers in the range 0 to 65536.

The **x** device control command has been extended.

x u *n* If *n* is 1, start underlining of spaces. If *n* is 0, stop underlining of spaces. This is needed for the **cu** request in nroff mode and is ignored otherwise.

Drawing Commands

The **D** drawing command has been extended. These extensions are not used by GNU pic if the **−n** option is given.

Df *n*\n Set the shade of gray to be used for filling solid objects to *n*; *n* must be an integer between 0 and 1000, where 0 corresponds solid white and 1000 to solid black, and values in between correspond to intermediate shades of gray. This applies only to solid circles, solid ellipses and solid polygons. By default, a level of 1000 is used. Whatever color a solid object has, it should completely obscure everything beneath it. A value greater than 1000 or less than 0 can also be used: this means fill with the shade of gray that is currently being used for lines and text. Normally this is black, but some drivers may provide a way of changing this.

The corresponding **\D'f**...**'** command shouldn't be used since its argument is always rounded to an integer multiple of the horizontal resolution which can lead to surprising results.

DC *d*\n Draw a solid circle with a diameter of *d* with the leftmost point at the current position.

DE *dx dy*\n
Draw a solid ellipse with a horizontal diameter of *dx* and a vertical diameter of *dy* with the leftmost point at the current position.

Dp $dx_1\ dy_1\ dx_2\ dy_2 \cdots$ .lf 3501
$dx_n\ dy_n$\n Draw a polygon with, for $i = 1, \ldots, n+1$, the i-th vertex at the current position $+ \sum_{j=1}^{i-1} (dx_j, dy_j)$. At the moment, GNU pic only uses this command to generate triangles and rectangles.

DP $dx_1\ dy_1\ dx_2\ dy_2\ \cdots$ .lf 3513
$dx_n\ dy_n$\n Like **Dp** but draw a solid rather than outlined polygon.

Dt *n*\n Set the current line thickness to *n* machine units. Traditionally Unix troff drivers use a line thickness proportional to the current point size; drivers should continue to do this if no **Dt** command has been given, or if a **Dt** command has been given with a negative value of *n*. A zero value of *n* selects the smallest available line thickness.

A difficulty arises in how the current position should be changed after the execution of these commands. This is not of great importance since the code generated by GNU pic does not depend on this. Given a drawing command of the form

\D'*c* $x_1\ y_1\ x_2\ y_2\ \cdots$ .lf 3546 $x_n\ y_n$**'**

where *c* is not one of **c**, **e**, **l**, **a**, or **~**, Unix troff treats each of the x_i as a horizontal quantity, and each of the y_i as a vertical quantity and assumes that the width of the drawn object is $\sum_{i=1}^{n} x_i$, and that the height is $\sum_{i=1}^{n} y_i$. (The assumption about the height can be seen by examining the **st** and **sb** registers after using such a **D** command in a **\w** escape sequence). This rule also holds for all the original drawing commands with the exception of **De**. For the sake of compatibility GNU troff also follows this rule, even though it produces an ugly result in the case of the **Dt** and **Df**, and, to a lesser extent, **DE** commands. Thus after executing a **D** command of the form

D*c* $x_1\ y_1\ x_2\ y_2\ \cdots$ .lf 3590 $x_n\ y_n$\n

the current position should be increased by $(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i)$.

Another set of extensions is

DFc *cyan magenta yellow*\n
DFd\n
DFg *gray*\n
DFk *cyan magenta yellow black*\n
DFr *red green blue*\n
Set the color components of the filling color similar to the **m** commands above.

The current position isn't changed by those colour commands (contrary to **Df**).

Device Control Commands

There is a continuation convention which permits the argument to the **x X** command to contain newlines: when outputting the argument to the **x X** command, GNU troff follows each newline in the argument with a **+** character (as usual, it terminates the entire argument with a newline); thus if the line after the line containing the **x X** command starts with **+**, then the newline ending the line containing the **x X** command should be treated as part of the argument to the **x X** command, the **+** should be ignored, and the part of the line following the **+** should be treated like the part of the line following the **x X** command.

The first three output commands are guaranteed to be:

x T *device*
x res *n h v*
x init

INCOMPATIBILITIES

In spite of the many extensions, groff has retained compatibility to classical troff to a large degree. For the cases where the extensions lead to collisions, a special compatibility mode with the restricted, old functionality was created for groff.

Groff Language

groff provides a **compatibility mode** that allows the processing of roff code written for classical **troff** or for other implementations of roff in a consistent way.

Compatibility mode can be turned on with the **−C** command-line option, and turned on or off with the **.cp** request. The number register **\n(.C** is 1 if compatibility mode is on, 0 otherwise.

This became necessary because the GNU concept for long names causes some incompatibilities. *Classical troff* interprets

.dsabcd

as defining a string **ab** with contents **cd**. In*gr off* mode, this is considered as a call of a macro named **dsabcd**.

Also *classical troff* interprets ***[** or **\n[** as references to a string or number register called **[** while *groff* takes this as the start of a long name.

In *compatibility mode*, groff interprets these things in the traditional way; so long names are not recognized.

On the other hand, groff in*GNU native mode* does not allo w to use the single-character escapes \\ (backslash), \| (vertical bar), \^ (caret), **\&** (ampersand), **\{** (opening brace), **\}** (closing brace), '\ ' (space),\' (single quote), \` (backquote), \− (minus), _ (underline), **\!** (bang), **\%** (percent), and **\c** (character c) in names of strings, macros, diversions, number registers, fonts or environments, whereas *classical troff* does.

The **\A** escape sequence can be helpful in avoiding these escape sequences in names.

Fractional point sizes cause one noteworthy incompatibility. In*classical tr off*, the **ps** request ignores scale indicators and so

.ps 10u

sets the point size to 10 points, whereas in groff native mode the point size is set to 10 scaled points.

In *groff*, there is a fundamental difference between unformatted input characters, and formatted output characters (glyphs). Everything that affects how a glyph is output is stored with the glyph; once a glyph has been constructed it is unaffected by any subsequent requests that are executed, including the **bd**, **cs**, **tkf**, **tr**, or **fp** requests.

Normally glyphs are constructed from input characters at the moment immediately before the glyph is added to the current output line. Macros, diversions and strings are all, in fact, the same type of object; they contain lists of input characters and glyphs in any combination.

Special characters can be both; before being added to the output, they act as input entities, afterwards they denote glyphs.

A glyph does not behave like an input character for the purposes of macro processing; it does not inherit any of the special properties that the input character from which it was constructed might have had. The following example makes things clearer.

```
.di x
\\\\
.br
.di
.x
```

With *GNU troff* this is printed as \\. So each pair of input backslashes '\\' is turned into a single output backslash glyph '\' and the resulting output backslashes are not interpreted as escape characters when they are reread.

Classical troff would interpret them as escape characters when they were reread and would end up printing a single backslash '\'.

In GNU, the correct way to get a printable version of the backslash character '\' is the **\(rs** escape sequence, but classical troff does not provide a clean feature for getting a non-syntactical backslash. A close method is the printable version of the current escape character using the **\e** escape sequence; this works if the current escape character is not redefined. It works in both GNU mode and compatibility mode, while dirty tricks like specifying a sequence of multiple backslashes do not work reliably; for the different handling in diversions, macro definitions, or text mode quickly leads to a confusion about the necessary number of backslashes.

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either the traditional **\!** transparent output facility or the new **\?** escape sequence can be used.

Intermediate Output

The groff intermediate output format is in a state of evolution. So far it has some incompatibilities, but it is intended to establish a full compatibility to the classical troff output format. Actually the following incompatibilities exist:

- The positioning after the drawing of the polygons conflicts with the classical definition.
- The intermediate output cannot be rescaled to other devices as classical 'device-independent' troff did.

AUTHORS

This document was written by James Clark ⟨jjc@jclark.com⟩ and modified by Werner Lemberg ⟨wl@gnu.org⟩ and Bernd Warken ⟨groff−bernd.warken−72@web.de⟩.

SEE ALSO

Groff: The GNU Implementation of troff, by Trent A. Fisher and Werner Lemberg, is the primary *groff* manual. You can browse it interactively with "info groff".

groff(1)
A list of all documentation around *groff*.

groff(7)
A description of the *groff* language, including a short, but complete reference of all predefined requests, registers, and escapes of plain *groff*. From the command line, this is called using

```
man 7 groff
```

roff(7) A survey of *roff* systems, including pointers to further historical documentation.

[*CSTR #54*]
The *Nroff/Troff User's Manual* by *J. F. Ossanna* of 1976 in the revision of *Brian Kernighan* of 1992, being the classical troff documentation ⟨http://cm.bell−labs.com/cm/cs/cstr/54.ps.gz⟩.