Name

groff_diff – differences between GNU *roff* and AT&T *troff*

Description

The GNU *roff* text processing system, *groff*, is an extension of AT&T *troff*, the typesetting system originating in Unix systems of the 1970s. *groff* removes many arbitrary limitations and adds features, both to the input language and to the page description language output by the *troff* formatter. Differences arising from *groff*'s implementation of AT&T *troff* features are also noted. See *roff* (7) for background.

Language

GNU *troff* features identifiers of arbitrary length; supports color output, non-integral type sizes, and user-defined characters; adds more conditional expression operators; recognizes additional scaling units and numeric operators; enables general file I/O (in "unsafe mode" only); and exposes more formatter state.

Long names

GNU *troff* introduces many new requests; with three exceptions (**cp**, **do**, **rj**), they have names longer than two characters. The names of registers, fonts, strings/macros/diversions, environments, special characters, streams, and colors can be of any length. Anywhere AT&T *troff* supports a parameterized escape sequence that uses an opening parenthesis "(" to introduce a two-character argument, *groff* supports a square-bracketed form "[]" where the argument within can be of arbitrary length.

Font families, abstract styles, and translation

GNU *troff* can group text typefaces into *families* containing each of the styles "**R**", "**I**", "**B**", and "**BI**". So that a document need not be coupled to a specific font family, an output device can associate a style in the abstract sense with a mounting position. Thus the default family can be combined with a style dynamically, producing a *resolved font name.* A document can *translate,* or remap, fonts with the **ftr** request.

Applying the requests **cs**, **bd**, **tkf**, **uf**, or **fspecial** to an abstract style affects the member of the default family corresponding to that style. The default family can be set with the **fam** request or **−f** command-line option. The **styles** directive in the output device's *DESC* file controls which mounting positions (if any) are initially associated with abstract styles rather than fonts, and the **sty** request can update this association.

Colors

groff supports color output with a variety of color spaces and up to 16 bits per channel. Some devices, particularly terminals, may be more limited. When color support is enabled, two colors are current at any given time: the *stroke color,* with which glyphs, rules (lines), and geometric figures are drawn, and the *fill color,* which paints the interior of filled geometric figures. The **color**, **defcolor**, **gcolor**, and **fcolor** requests; **\m** and **\M** escape sequences; and **.color**, **.m**, and **.M** registers exercise color support.

Fractional type sizes and new scaling units

AT&T *troff* interpreted all type size measurements in points. Combined with integer arithmetic, this design choice made it impossible to support, for instance, ten and a half-point type. In GNU *troff*, an output device can select a scaling factor that subdivides a point into "scaled points". A type size expressed in scaled points can thus represent a non-integral type size.

A *scaled point* is equal to 1/*sizescale* points, where *sizescale* is specified in the device description file, *DESC*, and defaults to 1; see *groff_font*(5). Requests and escape sequences in GNU *troff* interpret arguments that represent a type size in points, which the formatter multiplies by *sizescale* and converts to an integer. Arguments treated in this way comprise those to the escape sequences **\H** and **\s**, to the request **ps**, the third argument to the **cs** request, and the second and fourth arguments to the **tkf** request. Scaled points may be specified explicitly with the **z** scaling unit. In GNU *troff*, the register **\n[.s]** can interpolate a non-integral type size. The register **\n[.ps]** interpolates the type size in scaled points.

For example, if *sizescale* is 1000, then a scaled point is one thousandth of a point. Consequently, "**.ps 10.5**" is synonymous with "**.ps 10.5z**"; both set the type size to 10,500 scaled points, or 10.5 points.

It makes no sense to use the "**z**" scaling unit in a numeric expression whose default scaling unit is neither "**u**" nor "**z**", so GNU *troff* disallows this. Similarly, it is nonsensical to use a scaling unit other than "**z**" or "**u**" in a numeric expression whose default scaling unit is "**z**", so GNU *troff* disallows this as well.

Another new scaling unit, "**s**", multiplies by the number of basic units in a scaled point. Thus, "**\n[.ps]s**" is equal to "**1m**" by definition. Do not confuse the "**s**" and "**z**" scaling units.

Output devices may be limited in the type sizes they can employ. The **.s** and **.ps** registers represent the type size as selected by the output driver as it understands a device's capability. The last *requested* type size is interpolated in scaled points by the read-only register **.psr** and in points as a decimal fraction by the read-only string-valued register **.sr**. Both are associated with the environment. For example, if a type size of 10.95 points is requested, and the nearest size permitted by a **sizes** request (or by the **sizes** or **sizescale** directives in the device's *DESC* file) is 11 points, the output driver uses the latter value.

A further two new measurement units available in *groff* are "**M**", which indicates hundredths of an em, and "**f**", which multiplies by 65,536. The latter provides convenient fractions for color definitions with the **defcolor** request. For example, 0.5f equals 32768u.

Numeric expressions

GNU *troff* permits spaces in a numeric expression within parentheses, and offers three new operators.

*e1***>?***e2* Interpolate the greater of *e1* and *e2*.

*e1***<?***e2* Interpolate the lesser of *e1* and *e2*.

(*c***;***e***)** Evaluate *e* using *c* as the default scaling unit, ignoring scaling units in *e* if *c* is empty.

Conditional expressions

More conditions can be tested with the "**if**" and **ie** requests, as well as the new "**while**" request.

c *chr* True if a character *chr* is available, where *chr* is an ordinary character (Unicode basic Latin excluding control characters and the space), a special character, or **\N'***index***'**.

d *nam* True if a string, macro, diversion, or request *nam* is defined.

F *fnt* True if a font *fnt* is available; *fnt* can be an abstract style or a font name. *fnt* is handled as if it were accessed with the **ft** request (that is, abstract styles and font translation are applied), but *fnt* cannot be a mounting position, and no font is mounted.

m *col* True if a color *col* is defined.

r *reg* True if a register *reg* is defined.

S *sty* True if a style *sty* is registered. Font translation applies.

v Always false. This condition is for compatibility with certain other *troff* implementations only. (This refers to *vtroff*, a translator that would convert the C/A/T output from early-vintage AT&T *troff* to a form suitable for Versatec and Benson-Varian plotters.)

Drawing commands

GNU *troff* offers drawing commands to create filled circles and ellipses, and polygons. Stroked (outlined) objects are drawn with the stroke color and filled (solid) ones shaded with the fill color. These are independent properties; if you want a filled, stroked figure, you must draw the same figure twice using each drawing command. A filled figure is always smaller than a stroked one because the former is drawn only within its defined area, whereas strokes have a line thickness (set with another new drawing command, **\D't'**).

Escape sequences

groff introduces several new escape sequences and extends the syntax of a few AT&T *troff* escape sequences (namely, **\D**, **\f**, **\k**, **\n**, **\s**, **\$**, and *****). In the following list, escape sequences are collated alphabetically at first, and then by symbol roughly in Unicode code point order.

\A'*anything***'**

Interpolate 1 if *anything* is a valid identifier, and 0 otherwise. Because invalid input characters are removed, invalid identifiers are empty or contain spaces, tabs, or newlines. You can employ **\A** to validate a macro argument before using it to construct another escape sequence or identifier.

\B'*anything***'**

Interpolate 1 if *anything* is a valid numeric expression, and 0 otherwise. You might use **\B** along with the "**if**" request to filter out invalid macro arguments.

\D'C *d*' Draw filled circle of diameter *d* with its leftmost point at the drawing position.

\D'E *h v*'
Draw filled ellipse with *h* and *v* as the axes and the leftmost point at the drawing position.

\D'p *h1 v1 . . . hn vn*'
Draw polygon with vertices at drawing position and each point in sequence. GNU *troff* closes the polygon by drawing a line from (*hn*, *vn*) back to the initial drawing position; DWB and Heirloom *troff*s do not. Afterward, the drawing position is left at (*hn*, *vn*).

\D'P *h1 v1 . . . hn vn*'
As **\D'p'**, but the polygon is filled.

\D't *n*' Set line thickness of geometric objects to to *n* basic units. A zero *n* selects the minimal supported thickness. A negative *n* selects a thickness proportional to the type size; this is the default.

\E Embed an escape character that is not interpreted in copy mode (compare with **\a** and **\t**). You can use it to ease the writing of nested macro definitions. It is also convenient to define strings containing escape sequences that need to work when used in copy mode (for example, as macro arguments), or which will be interpolated at varying macro nesting depths.

\f[*font***]** Select *font*, which may be a mounting position, abstract style, or font name, to choose the typeface. **\f[]** and **\fP** are synonyms; we recommend the former.

\F*f*
\F(*fm*
\F[*family***]**
Select default font family. **\F[]** makes the previous font family the default. **\FP** is unlike **\fP**; it selects font family "P" as the default. See the **fam** request below.

\k(*rg*
\k[*reg***]** Mark horizontal drawing position in two-character register name *rg* or arbitrary register name *reg*.

\m*c*
\m(*cl*
\m[*col***]** Set the stroke color. **\m[]** restores the previous stroke color, or the default if there is none.

\M*c*
\M(*cl*
\M[*col***]** Set the fill color. **\M[]** restores the previous fill color, or the default if there is none.

\n[*reg***]** Interpolate register *reg*.

\O*n*
\O[*n***]** Suppress *troff* output of glyphs and geometric objects. The sequences **\O2**, **\O3**, **\O4**, and **\O5** are intended for internal use by *grohtml*(1).

\O0
\O1 Disable and enable, respectively, the emission of glyphs and geometric objects to the output driver, provided that this sequence occurs at the outermost suppression level (see **\O3** and **\O4**). Horizontal motions corresponding to non-overstruck glyph widths still occur. These sequences also reset the registers **opminx**, **opminy**, **opmaxx**, and **opmaxy** to −1. These four registers mark the top left and bottom right hand corners of a box encompassing all written or drawn output.

\O2 At the outermost suppression level, enable emission of glyphs and geometric objects, and write to the standard error stream the page number and values of the four aforementioned registers encompassing glyphs written since the last interpolation of a **\O** sequence, as well as the page offset, line length, image file name (if any), horizontal and vertical device motion quanta, and input file name. Numeric values are in basic units.

\O3
\O4 Begin and end a nested suppression level, respectively. *grohtml* uses this mechanism to create images of output preprocessed with *pic*, *eqn*, and *tbl*. At startup, *troff* is at the outermost suppression level. *pre-grohtml* generates these sequences when processing the document, using *troff* with the **ps** output device, Ghostscript, and the PNM tools to produce images in PNG format. These sequences start a new page if the device is not **html** or **xhtml**, to reduce the number of images crossing a page boundary.

\O5[*Pfile***]**
At the outermost suppression level, write the name *file* to the standard error stream at position *P*, which must be one of **l**, **r**, **c**, or **i**, corresponding to left, right, centered, and inline alignments within the document, respectively. *file* is is a name associated with the production of the next image.

\R'*name* ±*n***'**
Synonymous with "**.nr** *name* ±*n*".

\s[±*n***]**
\s±[*n***]**
\s'±*n***'**
\s±'*n***'** Set the type size to, or increment or decrement it by, *n* scaled points.

\V*e*
\V(*ev*
\V[*env***]** Interpolate contents of the environment variable *env*, as returned by *getenv*(3). **\V** is interpreted even in copy mode.

\X'*anything***'**
Within **\X** arguments, the escape sequences **\&**, **\)**, **\%**, and **\:** are ignored; *****space* and **\~** are converted to single space characters; and **** is reduced to ****. So that the basic Latin subset of the Unicode character set (that is, ISO 646:1991-IRV or, popularly, "US-ASCII") can be reliably encoded in *anything,* the special character escape sequences **\-**, **\[aq]**, **\[dq]**, **\[ga]**, **\[ha]**, **\[rs]**, and **\[ti]** are mapped to basic Latin characters; see *groff_char*(7). For this transformation, character translations and definitions are ignored. Other escape sequences are not supported.

If the **use_charnames_in_special** directive appears in the output device's *DESC* file, the use of special character escape sequences is *not* an error; they are simply output verbatim (with the exception of the seven mapped to Unicode basic Latin characters, discussed above). **use_charnames_in_special** is currently employed only by *grohtml*(1).

\Y*m*
\Y(*ma*
\Y[*mac***]**
Interpolate a macro as a device control command. This is similar to **\X'*[***mac***]'**, except the contents of *mac* are not interpreted, and *mac* can be a macro and thus contain newlines, whereas the argument to **\X** cannot. This inclusion of newlines requires an extension to the AT&T *troff* output format, and will confuse postprocessors that do not know about it.

\Z'*anything***'**
Save the drawing position, format *anything*, then restore it. Tabs and leaders in the argument are ignored with an error diagnostic.

\# Everything up to and including the next newline is ignored. This escape sequence is interpreted even in copy mode. **\#** is like **\"**, except that **\"** does not ignore a newline; the latter therefore cannot be used by itself for a whole-line comment—it leaves a blank line on the input stream.

\$0 Interpolate the name by which the macro being interpreted was called. In GNU *troff* this name can vary; see the **als** request.

\$(*nn*

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

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

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

\$^ In a macro or string definition, interpolate the catenation of all arguments constructed in a form suitable for passage to the **ds** request.

\) Interpolate a *transparent* dummy character—one that is ignored by end-of-sentence detection. It behaves as **\&**, except that **\&** is treated as letters and numerals normally are after ".", "?", and "!"; **\&** cancels end-of-sentence detection, and **\)** does not.

[string* [*arg* ...]]
Interpolate *string,* passing it *arg* ... as arguments.

\/ Apply an *italic correction*: modify the spacing of the preceding glyph so that the distance between it and the following glyph is correct if the latter is of upright shape. For example, if an italic "f" is followed immediately 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 **\/** between them produces *f*) and avoids this problem. Use this escape sequence whenever an oblique glyph is immediately followed by an upright glyph without any intervening space.

\, Apply a *left italic correction*: modify the spacing of the following glyph so that the distance between it and the preceding glyph is correct if the latter is of upright shape. For example, if a roman left parenthesis is immediately followed by an italic "f", then in many fonts the bottom left portion of the "f" overlaps the bottom of the left parenthesis, producing (*f*, which is ugly. Inserting **\,** between them produces (*f* and avoids this problem. Use this escape sequence whenever an upright glyph is followed immediately by an oblique glyph without any intervening space.

\: Insert a non-printing break point. That is, a word can break there, but the soft hyphen character does not mark the break point if it does (in contrast to "**\%**"). This escape sequence is an input word boundary, so the remainder of the word is subject to hyphenation as normal.

\?*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 becomes an 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**.

\[*char***]** Typeset the special character *char*.

\[*base-char combining-component* . . .**]**
Typeset a composite glyph consisting of *base-char* overlaid with one or more *combining-component*s. For example, "**\[A ho]**" is a capital letter "A" with a "hook accent" (ogonek). See the **composite** request below; *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual, for details of composite glyph name construction; and *groff_char*(7) for a list of components used in composite glyph names.

\~ Insert an unbreakable space that is adjustable like an ordinary space. It is discarded from the end of an output line if a break is forced.

Restricted requests

To mitigate risks from untrusted input documents, the **pi** and **sy** requests are disabled by default. *troff*(1)'s **−U** option enables the formatter's "unsafe mode", restoring their function (and enabling additional *groff* extension requests, **open**, **opena**, and **pso**).

New requests

.aln *new old*
Create alias *new* for existing register named *old*, causing the names to refer to the same stored value. If *old* is undefined, a warning in category "**reg**" is generated and the request is ignored. To remove a register alias, invoke **rr** on its name. A register's contents do not become inaccessible until it has no more names.

.als *new old*
Create alias *new* for existing request, string, macro, or diversion named *old*, causing the names to refer to the same stored object. If *old* is undefined, a warning in category "**mac**" is produced, and the request is ignored. The "**am**", "**as**", **da**, **de**, **di**, and **ds** requests (together with their variants) create a new object only if the name of the macro, diversion, or string is currently undefined or if it is defined as a request; normally, they modify the value of an existing object. To remove an alias, invoke **rm** on its name. The object itself is not destroyed until it has no more names.

When a request, macro, string, or diversion is aliased, redefinitions and appendments "write through" alias names. To replace an alias with a separately defined object, you must use the **rm** request on its name first.

.am1 *name* [*end-name*]
As "**am**", but compatibility mode is disabled while the appendment to *name* is interpreted: a "compatibility save" token is inserted at its beginning, and a "compatibility restore" token at its end. As a consequence, the requests "**am**", **am1**, **de**, and **de1** can be intermixed freely since the compatibility save/restore tokens affect only the parts of the macro populated by **am1** and **de1**.

.ami *name* [*end-name*]
Append to macro indirectly. See **dei** below.

.ami1 *name* [*end-name*]
As **ami**, but compatibility mode is disabled during interpretation of the appendment.

.as1 *name* [*contents*]
As "**as**", but compatibility mode is disabled while the appendment to *name* is interpreted: a "compatibility save" token is inserted at the beginning of *contents*, and a "compatibility restore" token after it. As a consequence, the requests "**as**", **as1**, **ds**, and **ds1** can be intermixed freely since the compatibility save/restore tokens affect only the portions of the strings populated by **as1** and **ds1**.

.asciify *div*
Unformat the diversion *div* in a way such that Unicode basic Latin (ASCII) characters, characters translated with the **trin** request, space characters, and some escape sequences, that were formatted in the diversion *div* are treated like ordinary input characters when *div* is reread. Doing so can be useful in conjunction with the **writem** request. **asciify** can be also used for gross hacks; for example, the following sets register **n** to 1.

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

asciify cannot return all items in a diversion to their source equivalent: nodes such as those produced by **\N[**...**]** will remain nodes, so the result cannot be guaranteed to be a pure string. See section "Copy mode" in *groff*(7). Glyph parameters such as the type face and size are not preserved; use **unformat** to achieve that.

.backtrace

Write backtrace of input stack to the standard error stream. See the **−b** option of *troff*(1).

.blm [*name*]

Set a blank line macro (trap). If a blank line macro is thus defined, *groff* executes *macro* when a blank line is encountered in the input file, instead of the usual behavior. A line consisting only of spaces is also treated as blank and subject to this trap. If no argument is supplied, the default blank line behavior is (re-)established.

.box [*name*]

.boxa [*name*]

Divert (or append) output to *name,* similarly to the **di** and **da** requests, respectively. Any pending output line is *not* included in the diversion. Without an argument, stop diverting output; any pending output line inside the diversion is discarded.

.break Exit a "**while**" loop. Do not confuse this request with a typographical break or the **br** request. See "**continue**".

.brp Break and adjust line; this is the AT&T *troff* escape sequence **\p** in request form.

.cflags *n c1 c2* ...

Assign properties encoded by the number *n* to characters *c1*, *c2*, and so on. Ordinary and special characters have certain associated properties. (Glyphs don't: to GNU *troff*, like AT&T device-independent *troff*, a glyph is an identifier corresponding to a rectangle with some metrics; see *groff_font*(5).) The first argument is the sum of the desired flags and the remaining arguments are the characters to be assigned those properties. Spaces between the *cn* arguments are optional. Any argument *cn* can be a character class defined with the **class** request rather than an individual character.

The non-negative integer *n* is the sum of any of the following. Some combinations are nonsensical, such as "**33**" (1 + 32).

1 Recognize the character as ending a sentence if followed by a newline or two spaces. Initially, characters "**.?!**" have this property.

2 Enable breaks before the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, no characters have this property.

4 Enable breaks after the character. A line is not broken at a character with this property unless the characters on each side both have non-zero hyphenation codes. This exception can be overridden by adding 64. Initially, characters "**−\[hy]\[em]**" have this property.

8 Mark the glyph associated with this character as overlapping other instances of itself horizontally. Initially, characters "**\[ul]\[rn]\[ru]\[radicalex]\[sqrtex]**" have this property.

16 Mark the glyph associated with this character as overlapping other instances of itself vertically. Initially, the character "**\[br]**" has this property.

32 Mark the character as transparent for the purpose of end-of-sentence recognition. In other words, 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. This is the same as having a zero space factor in TEX. Initially, characters "**' ")]*\[dg]\[dd]\[rq]\[cq]**" have this property.

64 Ignore hyphenation codes of the surrounding characters. Use this value in combination with values 2 and 4. Initially, no characters have this property.

For example, if you need an automatic break point after the en-dash in numeric ranges like "3000–5000", insert

```
.cflags 68 \[en]
```

into your document. However, this can lead to bad layout if done without thinking; in most situations, a better solution than changing the **cflags** value is inserting "**\:**" right after the hyphen at the places that really need a break point.

The remaining values were implemented for East Asian language support; those who use alphabetic scripts exclusively can disregard them.

128 Prohibit a break before the character, but allow a break after the character. This works only in combination with values 256 and 512 and has no effect otherwise. Initially, no characters have this property.

256 Prohibit a break after the character, but allow a break before the character. This works only in combination with values 128 and 512 and has no effect otherwise. Initially, no characters have this property.

512 Allow a break before or after the character. This works only in combination with values 128 and 256 and has no effect otherwise. Initially, no characters have this property.

In contrast to values 2 and 4, the values 128, 256, and 512 work pairwise. If, for example, the left character has value 512, and the right character 128, no break will be automatically inserted between them. If we use value 6 instead for the left character, a break after the character can't be suppressed since the neighboring character on the right doesn't get examined.

.char *c contents*

Define the ordinary or special character *c* as *contents*, which can be empty. More precisely, **char** defines a *groff* object (or redefines an existing one) that is accessed with the name *c* on input, and produces *contents* on output. Every time *c* is to be formatted, *contents* 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 *contents* is processed. Any emboldening, constant spacing, or track kerning is applied to this object as a whole, not to each character in *contents*.

An object defined by this request can be used just like a glyph provided by the output device. In particular, other characters can be translated to it with the **tr** request; it can be made the tab or leader fill character with the **tc** and **lc** requests; sequences of it can be drawn with the **\l** and **\L** escape sequences; and, if the **hcode** request is used on *c*, it is subject to automatic hyphenation.

To prevent infinite recursion, occurrences of *c* within its own definition are treated normally (as if it were not being defined with **char**). The **tr** and **trin** requests take precedence if **char** both apply to *c*. A character definition can be removed with the **rchar** request.

.chop *object*

Remove the last character from the macro, string, or diversion *object*. This is useful for removing the newline from the end of a diversion that is to be interpolated as a string. This request can be used repeatedly on the same *object*; see section "gtroff Internals" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual, for discussion of nodes inserted by *groff*.

.class *name c1 c2 . . .*

Define a character class (or simply "class") *name* comprising the characters or range expressions *c1*, *c2*, and so on.

A class thus defined can then be referred to in lieu of listing all the characters within it. Currently, only the **cflags** request can handle references to character classes.

In the request's simplest form, each *cn* is a character (or special character).

```
.class [quotes] ' \[aq] \[dq] \[oq] \[cq] \[lq] \[rq]
```

Since class and special character names share the same name space, we recommend starting and ending the class name with "**[**" and "**]**", respectively, to avoid collisions with existing character names defined by *groff* or the user (with **char** and related requests). This practice applies the presence of "**]**" in the class name to prevent the usage of the special character escape form "**\[**. . .**]**", thus you must use the **\C** escape to access a class with such a name.

You can also use a character range expression consisting of a start character followed by "**–**" and then an end character. Internally, GNU *troff* converts these two character names to Unicode code points (according to the *groff* glyph list [GGL]), which determine the start and end values of the range. If that fails, the class definition is skipped. Furthermore, classes can be nested.

```
.class [prepunct] , : ; > }
.class [prepunctx] \C'[prepunct]' \[u2013]-\[u2016]
```

The class "**[prepunctx]**" thus contains the contents of the class "**[prepunct]**" and characters in the range U+2013–U+2016.

If you want to include "**–**" in a class, it must be the first character value in the argument list, otherwise it gets misinterpreted as part of the range syntax.

It is not possible to use class names as end points of range definitions.

A typical use of the **class** request is to control line-breaking and hyphenation rules as defined by the **cflags** request. For example, to inhibit line breaks before the characters belonging to the "**[prepunctx]**" class defined in the previous example, you can write the following.

```
.cflags 2 \C'[prepunctx]'
```

.close *stream*

Close the stream named *stream*, invalidating it as an argument to the **write** request. See **open**.

.composite *c1 c2*

Map character name *c1* to character name *c2* when *c1* is a combining component in a composite glyph. Typically, this remaps a spacing glyph to a combining one.

.continue

Skip the remainder of a "**while**" loop's body, immediately starting the next iteration. See **break**.

.color *n*

If *n* is non-zero or missing, enable colors (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 they cause do not arise.

.defcolor *ident scheme color-component . . .*

Define a color named *ident. scheme* identifies a color space and determines the number of required *color-component*s; it must be one of "**rgb**" (three components), "**cmy**" (three components), "**cmyk**" (four components), or "**gray**" (one component). "**grey**" is accepted as a synonym of "**gray**". The color components can be encoded as a hexadecimal value starting with **#** or **##**. The former indicates that each component is in the range 0–255 (0–FF), the latter the range 0–65535 (0–FFFF). Alternatively, each color component can be specified as a decimal fraction in the range 0–1, interpreted using a default scaling unit of "**f**", which multiplies its value by 65,536 (but clamps it at 65,535).

Each output device has a color named "**default**", which cannot be redefined. A device's default stroke and fill colors are not necessarily the same.

.de1 *name* [*end-name*]
Define a macro to be interpreted with compatibility mode disabled. When *name* is called, compatibility mode enablement status is saved; it is restored when the call completes.

.dei *name* [*end-name*]
Define macro indirectly, with the name of the macro to be defined in string *name* and the name of the end macro terminating its definition in string *end-name*.

.dei1 *name* [*end-name*]
As **dei**, but compatibility mode is disabled while the definition of the macro named in string *name* is interpreted.

.device *anything*
Write *anything*, read in copy mode, to *troff* output as a device control command. An initial neutral double quote is stripped to allow the embedding of leading spaces.

.devicem *name*
Write contents of macro or string *name* to *troff* output as a device control command.

.do *name* [*arg* ...]
Interpret the string, request, diversion, or macro *name* (along with any arguments) with compatibility mode disabled. Compatibility mode is restored (only if it was active) when the *expansion* of *name* is interpreted; that is, the restored compatibility state applies to the contents of the macro, string, or diversion *name* as well as data read from files or pipes if *name* is any of the **so**, **soquiet**, **mso**, **msoquiet**, or **pso** requests.

For example,

```
.de mac1
FOO
..
.de1 mac2
groff
.mac1
..
.de mac3
compatibility
.mac1
..
.de ma
\\$1
..
.cp 1
.do mac1
.do mac2 \" mac2, defined with .de1, calls "mac1"
.do mac3 \" mac3 calls "ma" with argument "c1"
.do mac3 \[ti] \" groff syntax accepted in .do arguments
```

results in

```
FOO groff FOO compatibility c1 ~
```

as output.

.ds1 *name contents*
As **ds**, but compatibility mode is disabled while *name* is interpreted: a "compatibility save" token is inserted at the beginning of *contents*, and a "compatibility restore" token after it.

.ecr Restore the escape character saved with **ecs**, or set escape character to "\" if none has been saved.

.ecs Save the current escape character.

.evc *env*

Copy the properties of environment *env* to the current environment, except for the following data.

- a partially collected line, if present;
- the interruption status of the previous input line (due to use of the **\c** escape sequence);
- the count of remaining lines to center, to right-justify, or to underline (with or without underlined spaces)—these are set to zero;
- the activation status of temporary indentation;
- input traps and their associated data;
- the activation status of line numbering (which can be reactivated with "**.nm +0**"); and
- the count of consecutive hyphenated lines (set to zero).

.fam [*family*]

Set default font family to *family*. If no argument is given, the previous font family is selected, or the formatter's default family if there is none. The formatter's default font family is "T" (Times), but it can be overridden by the output device—see *groff_font*(5). The default font family is associated with the environment. See **\F**.

.fchar *c contents*

Define fallback character *c* as *contents*. The syntax of this request is the same as the **char** request; the difference is that a character defined with **char** hides a glyph with the same name in the selected font, whereas characters defined with **fchar** are checked only if *c* isn't found in the selected font. This test happens before special fonts are searched.

.fcolor *color*

Set the fill color to *color*. Without an argument, the previous fill color is selected.

.fschar *f c contents*

Define fallback special character *c* for font *f* as *contents*. A character defined by **fschar** is located after the list of fonts declared with **fspecial** is searched but before those declared with the "**special**" request.

.fspecial *f s1 s2* ...

When font *f* is selected, fonts *s1*, *s2*, ... are treated as special; that is, they are searched for glyphs not found in *f*. Any fonts specified in the "**special**" request are searched after *s1*, *s2*, and so on. Without *s* arguments, **fspecial** clears the list of fonts treated as special when *f* is selected.

.ftr *f g* Translate font *f* to *g*. Whenever a font named *f* is referred to in an **\f** escape sequence, in the **F** and **S** conditional expression operators, or in the **ft**, **ul**, **bd**, **cs**, **tkf**, **special**, **fspecial**, **fp**, or **sty** requests, font *g* is used. If *g* is missing or identical 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 resolved font name, not an abstract style.

.gcolor *color*

Set the stroke color to *color*. Without an argument, the previous stroke color is selected.

.hcode *c1 code1* [*c2 code2*] ...

Set the hyphenation code of character *c1* to *code1*, that of *c2* to *code2*, and so on. A hyphenation code must be an ordinary character (not a special character escape sequence) other than a digit. The request is ignored if given no arguments.

For hyphenation to work, hyphenation codes must be set up. At startup, *groff* assigns hyphenation codes to the letters "a–z" (mapped to themselves), to the letters "A–Z" (mapped to "a–z"), and zero to all other characters. Normally, hyphenation patterns contain only lowercase letters which should be applied regardless of case. In other words, they assume that the words "ABBOT" and "Abbot" should be hyphenated exactly as "abbot" is. **hcode** extends this principle to letters

outside the Unicode basic Latin alphabet; without it, words containing such letters won't be hyphenated properly even if the corresponding hyphenation patterns contain them.

.hla *lang*

Set the hyphenation language to *lang*. Hyphenation exceptions specified with the **hw** request and hyphenation patterns and exceptions specified with the **hpf** and **hpfa** requests are associated with the hyphenation language. The **hla** request is usually invoked by a localization file, which is in turn loaded by the *troffrc* or *troffrc−end* file; see the **hpf** request below. The hyphenation language is associated with the environment.

.hlm [*n*]

Set the maximum number of consecutive hyphenated lines to *n*. If *n* is negative, there is no maximum. If omitted, *n* is −1. This value is associated with the environment. Only lines output from a given environment count towards the maximum associated with that environment. Hyphens resulting from **\%** are counted; explicit hyphens are not.

.hpf *pattern-file*

Read hyphenation patterns from *pattern-file*. This file is sought in the same way that macro files are with the **mso** request or the **−m***name* command-line option to *groff* (1) and *troff* (1).

The *pattern-file* should have the same format as (simple) TEX pattern files. 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.
- "Digraphs" like **\$** are not supported.
- "**^^***xx*" (where each *x* is 0–9 or a–f) and **^^***c* (character *c* in the code point range 0–127 decimal) are recognized; other uses of **^** cause an error.
- No macro expansion is performed.
- **hpf** checks for the expression **\patterns{**...**}** (possibly with whitespace before or 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 (but the "**%**" character is still recognized as the start of a comment).

Use the **hpfcode** request (see below) to map the encoding used in hyphenation pattern files to *groff*'s input encoding.

The set of hyphenation patterns is associated with the hyphenation language set by the **hla** request. The **hpf** request is usually invoked by a localization file loaded by the *troffrc* file. By default, *troffrc* loads the localization file for English. (As of *groff* 1.23.0, localization files for Czech (*cs*), German (*de*), English (*en*), French (*fr*), Japanese (*ja*), Swedish (*sv*), and Chinese (*zh*) exist.) For Western languages, the localization file sets the hyphenation mode and loads hyphenation patterns and exceptions.

A second call to **hpf** (for the same language) replaces the old patterns with the new ones.

Invoking **hpf** causes an error if there is no hyphenation language.

If no **hpf** request is specified (either in the document, in a file loaded at startup, or in a macro package), GNU *troff* won't automatically hyphenate at all.

.hpfa *pattern-file*

As **hpf**, except that the hyphenation patterns and exceptions from *pattern-file* are appended to the patterns already applied to the hyphenation language of the environment.

.hpfcode *a b* [*c d*] . . .
Define mapping values for character codes in pattern files. This is an older mechanism no longer used by *groff*'s own macro files; for its successor, see **hcode** above. **hpf** or **hpfa** apply the mapping after reading or appending to the active list of patterns. Its arguments are pairs of character codes—integers from 0 to 255. The request maps character code *a* to code *b*, code *c* to code *d*, and so on. Character codes that would otherwise be invalid in *groff* can be used. By default, every code maps to itself except those for letters "A" to "Z", which map to those for "a" to "z".

.hym [*length*]
Set the (right) hyphenation margin to *length*. If the adjustment mode is not "**b**" or "**n**", the line is not hyphenated if it is shorter than *length*. Without an argument, the default hyphenation margin is reset to its default value, 0. The default scaling unit is "**m**". The hyphenation margin is associated with the environment. A negative argument resets the hyphenation margin to zero, emitting a warning in category "**range**".

.hys [*hyphenation-space*]
Suppress hyphenation of the line in adjustment modes "**b**" or "**n**", if it can be justified by adding no more than *hyphenation-space* extra space to each inter-word space. Without an argument, the hyphenation space adjustment threshold is set to its default value, 0. The default scaling unit is "**m**". The hyphenation space adjustment threshold is associated with the current environment. A negative argument resets the hyphenation space adjustment threshold to zero, emitting a warning in category "**range**".

.itc *n name*
As "**it**", but lines interrupted with the **\c** escape sequence are not applied to the line count.

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

.length *reg anything*
Compute the number of characters in *anything* and return the count in the register *reg*. If *reg* doesn't exist, it is created. *anything* is read in copy mode.

```
.ds xxx abcd\h'3i'efgh
.length yyy \*[xxx]
\n[yyy]
14
```

.linetabs *n*
If *n* is non-zero or missing, enable line-tabs mode, otherwise disable it (the default). In this mode, tab stops are computed relative to the start of the pending output line, instead of the drawing position corresponding to the start of the input line. Line-tabs mode is a property of the environment.

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
```

whereas in line-tabs mode, the same input gives

```
a       b                       c
```

instead.

.lsm [*name*]
Set the leading space macro (trap) to *name*. If there are leading space characters on an input line, *name* is invoked in lieu of the usual *roff* behavior; the leading spaces are removed. The count of

leading spaces on an input line is stored in **\n[lsn]**, and the amount of corresponding horizontal motion in **\n[lss]**, irrespective of whether a leading space trap is set. When it is, the leading spaces are removed from the input line, and no motion is produced before calling *name*. If no argument is supplied, the default leading space behavior is (re-)established.

.mso *file*
: As "**so**", except that *file* is sought in the same directories as arguments to the *groff* (1) and *troff* (1) **−m** command-line option are (the "tmac path"). If the file name to be interpolated has the form *name***.tmac** and it isn't found, **mso** tries to include **tmac.***name* instead and vice versa. If *file* does not exist, a warning in category "**file**" is emitted and the request has no other effect.

.msoquiet *file*
: As **mso**, but no warning is emitted if *file* does not exist.

.nop *anything*
: Interpret *anything* as if it were an input line. **nop** resembles "**.if 1**"; it puts a break on the output if *anything* is empty. Unlike "**if**", it cannot govern conditional blocks. Its application is to maintain consistent indentation within macro definitions even when producing text lines.

.nroff
: Make the **n** conditional expression evaluate true and **t** false. See **troff**.

.open *stream file*
: Open *file* for writing and associate *stream* with it. See **write** and **close**.

.opena *stream file*
: As **open**, but if *file* exists, append to it instead of truncating it.

.output *contents*
: Emit *contents*, which are read in copy mode, to the formatter output; this is similar to **\!** used in the top-level diversion. An initial neutral double quote in *contents* is stripped to allow the embedding of leading spaces.

.pev
: Report the state of the current environment followed by that of all other environments to the standard error stream.

.pnr
: Write the names and values of all currently defined registers to the standard error stream.

.psbb *file*
: Get the bounding box of a PostScript image *file*. This file must conform to Adobe's Document Structuring Conventions; the request attempts to extract the bounding box values from a **%%BoundingBox** comment. After invocation, the *x* and *y* coordinates (in PostScript units) of the lower left and upper right corners can be found in the registers **\n[llx]**, **\n[lly]**, **\n[urx]**, and **\n[ury]**, respectively. If an error occurs, these four registers are set to zero.

.pso *command*
: As "**so**", except that input comes from the standard output stream of *command*.

.ptr
: Report the names and vertical positions of all page location traps to the standard error stream. Empty slots in the list are shown as well, because they can affect the visibility of subsequently planted traps.

.pvs ±*n*
: Set the post-vertical line spacing to *n*; default scaling unit is "**p**". With no argument, the post-vertical line space is set to its previous value.

  In GNU *troff*, the distance between text baselines consists of the extra pre-vertical line spacing set by the most negative **\x** argument on the pending output line, the vertical spacing (**vs**), the extra post-vertical line spacing set by the most positive **\x** argument on the pending output line, and the post-vertical line spacing set by this request.

.rchar *c* ...
: Remove definition of each ordinary or special character *c*, undoing the effect of a **char**, **fchar**, or **schar** request. Glyphs, which are defined by font description files, cannot be removed. Spaces and tabs may separate *c* arguments.

.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 *f c* ...
Remove each fallback special character *c* for font *f*. Spaces and tabs may separate *c* arguments. See **fschar**.

.rj [*n*] Right-align the next *n* input lines. Without an argument, right-align the next input line. **rj** implies "**.ce 0**", and **ce** implies "**.rj 0**".

.rnn *r1 r2*
Rename register *r1* to *r2*. If *r1* doesn't exist, the request is ignored.

.schar *c contents*
Define global fallback character *c* as *contents*. See **char**; the distinction is that a character defined with **schar** is located after the list of fonts declared with the **special** request but before any mounted special fonts.

.shc [*c*] Set the soft hyphen character, inserted when a word is hyphenated automatically or at a hyphenation character, to *c*. If *c* is omitted, the soft hyphen character is set to the default, **\[hy]**. If the selected glyph does not exist in the font in use at a potential hyphenation point, then the line is not broken at that point. Neither character definitions (**char** and similar) nor translations (**tr** and similar) are considered when assigning 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, arguments are shifted by 1. No effect otherwise.

.sizes *s1 s2* ... *sn* [**0**]
Set the available type sizes to *s1*, *s2*, ... *sn* scaled points. The list of sizes can be terminated by an optional "**0**". Each *si* can also be a range *m–n*. In contrast to the device description file directive of the same name (see *groff_font*(5)), the argument list can't extend over more than one line.

.soquiet *file*
As "**so**", but no warning is emitted if *file* does not exist.

.special *f* ...
Declare each font *f* as special, searching it for glyphs not found in the selected font. Without arguments, this list of special fonts is made empty.

.spreadwarn [*limit*]
Emit a **break** warning if the additional space inserted for each space between words in an output line adjusted to both margins with "**.ad b**" is larger than or equal to *limit*. A negative value is treated as zero; an absent argument toggles the warning on and off without changing *limit*. The default scaling unit is **m**. At startup, **spreadwarn** is inactive and *limit* is 3 m.

For example, "**.spreadwarn 0.2m**" causes a warning if **break** warnings are not suppressed and *troff* must add 0.2 m or more for each inter-word space in a line.

.stringdown *str*
.stringup *str*
Alter the string named *str* by replacing each of its bytes with its lowercase (**down**) or uppercase (**up**) version (if one exists). Special characters (see *groff_char*(7)) will often transform in the expected way due to the regular naming convention for accented characters. When they do not, use substrings and/or catenation.

```
.ds resume R\['e]sum\['e]\"
\*[resume]
.stringdown resume
\*[resume]
.stringup resume
\*[resume]
Résumé résumé RÉSUMÉ
```

.sty *n s* Associate abstract style *s* with font mounting position *n*.

.substring *string start* [*end*]

Replace the string named *string* with its substring bounded by the indices *start* and *end*, inclusively. The first character in the string has index 0. If *end* is omitted, it is implicitly set to the largest valid value (the string length minus one). Negative indices count backwards from the end of the string: the last character has index −1, the character before the last has index −2, and so on.

```
.ds xxx abcdefgh
.substring xxx 1 -4
\*[xxx]
bcde
.substring xxx 2
\*[xxx]
de
```

.tkf *f s1 n1 s2 n2*

Enable track kerning for font *f*. When the current font is *f* the width of every glyph is increased by an amount between *n1* and *n2*; when the current type 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 type 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 type size.

.tm1 *message*

As **tm** request, but strips a leading neutral double quote from *message* to allow the embedding of leading spaces.

.tmc *message*

As **tm1** request, but does not append a newline.

.trf *file* Transparently output the contents of file *file*. Each line is output as if preceded by **\!**; however, the lines are not subject to copy-mode interpretation. If the file does not end with a newline, then a newline is added. Unlike **cf**, *file* cannot contain characters that are invalid as input to GNU *troff*.

For example, you can define a macro *x* containing the contents of file *f*, using

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

.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 would 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 **t** conditional expression evaluate true and **n** false. See **nroff**.

.unformat *div*
Unformat the diversion *div*. Unlike **asciify**, **unformat** handles only tabs and spaces between words, the latter usually arising from spaces or newlines in the input. Tabs are treated as input tokens, and spaces become adjustable again. The vertical sizes of lines are not preserved, but glyph information (font, type size, space width, and so on) is retained.

.vpt *n* If *n* is non-zero or missing, enable vertical position traps (the default), otherwise disable them. Vertical position traps are those set by the **ch**, **wh**, and **dt** requests.

.warn [*n*]
Select the categories, or "types", of reported warnings. *n* is the sum of the numeric codes associated with each warning category that is to be enabled; all other categories are disabled. The categories and their associated codes are listed in section "Warnings" of *troff* (1). For example, "**.warn 0**" disables all warnings, and "**.warn 1**" disables all warnings except those about missing glyphs. If no argument is given, all warning categories are enabled.

.warnscale *si*
Set the scaling unit used in warnings to *si*. Valid values for *si* are **u**, **i** (the default), **c**, **p**, and **P**.

.while *cond-expr anything*
Evaluate the conditional expression *cond-expr*, and repeatedly execute *anything* unless and until *cond-expr* evaluates false. *anything,* which is often a conditional block, is referred to as the **while** request's *body.*

troff treats the body of a **while** request similarly to that of a **de** request (albeit one not read in copy mode), but stores it under an internal name and deletes it when the loop finishes. The operation of a macro containing a **while** request can slow significantly if the **while** body is large. Each time the macro is executed, the **while** body is parsed and stored again. An often better solution—and one that is more portable, since AT&T *troff* lacked the **while** request—is to instead write a recursive macro. It will be parsed only once (unless you redefine it). To prevent infinite loops, the default number of available recursion levels is 1,000 or somewhat less (because things other than macro calls can be on the input stack). You can disable this protective measure, or raise the limit, by setting the **slimit** register. See section "Debugging" below.

If a **while** body begins with a conditional block, its closing brace must end an input line.

The **break** and **continue** requests alter a **while** loop's flow of control.

.write *stream anything*
Write *anything* to *stream*, which must previously have been the subject of an **open** request, followed by a newline. *anything* is read in copy mode. An initial neutral double quote in *anything* is stripped to allow the embedding of leading spaces.

.writec *stream anything*
As **write**, but without a trailing newline.

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

Extended requests

.cf *file* In a diversion, embed an object which, when reread, will cause the contents of *file* to be copied verbatim to the output. In AT&T *troff*, the contents of *file* are immediately copied to the output regardless of whether a diversion is being written to; this behavior is so anomalous that it must be considered a bug.

.de *name* [*end-name*]
.am *name* [*end-name*]
.ds *name* [*contents*]
.as *name* [*contents*]
In compatibility mode, these requests behave similarly 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.

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

.ss *word-space-size* [*additional-sentence-space-size*]
A second argument sets the amount of additional space separating sentences on the same output line. If omitted, this amount is set to *word-space-size*. Both arguments are in twelfths of current font's space width (typically one-fourth to one-third em for Western scripts; see *groff_font*(5)). The default for both parameters is 12. Negative values are erroneous.

.ta [[*n1 n2 . . . nn*]**T** *r1 r2 . . . rn*]
groff supports an extended syntax to specify repeating tab stops after the "**T**" mark. These values are always taken as relative distances from the previous tab stop. This is the idiomatic way to specify tab stops at equal intervals in *groff*.

The syntax summary above instructs *groff* to set tabs at positions *n1*, *n2*, . . . , *nn*, then at *nn* + *r1*, *nn* + *r2*, . . . , *nn* + *rn*, then at *nn* + *rn* + *r1*, *nn* + *rn* + *r2*, . . . , *nn* + *rn* + *rn*, and so on.

New registers

GNU *troff* exposes more formatter state via many new read-only registers. Their names often correspond to the requests that affect them.

\n[.br] Within a macro call, interpolate 1 if the macro is called with the "normal" control character ("." by default), and 0 otherwise. This facility allows the reliable modification of requests. Using this register outside of a macro definition makes no sense.

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

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

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

\n[.ce] Interpolate number of input lines remaining to be centered.

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

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

\n[.cp] Within a "**do**" request, interpolate the saved value of compatibility mode (see **\n[.C]** above).

\n[.csk] Interpolate skew of last glyph added to the environment. The *skew* 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] Interpolate name of current environment. This is a string-valued register.

\n[.fam] Interpolate name of default font family. This is a string-valued register.

\n[.fn] Interpolate resolved name of the selected font. This is a string-valued register.

\n[.fp] Interpolate next free font mounting position.

\n[.g] Interpolate 1. Test with "**if**" or **ie** to check whether GNU *troff* is the formatter.

\n[.height] Interpolate font height. See **\H**.

\n[.hla] Interpolate hyphenation language of the environment. This is a string-valued register.

\n[.hlc] Interpolate count of immediately preceding consecutive hyphenated lines in the environment.

\n[.hlm] Interpolate maximum number of consecutive hyphenated lines allowed in the environment.

\n[.hy] Interpolate hyphenation mode of the environment.

\n[.hym] Inteprolate hyphenation margin of the environment.

\n[.hys] Interpolate hyphenation space adjustment threshold of the environment.

\n[.in] Interpolate indentation amount applicable to the pending output line.

\n[.int] Interpolate 1 if the previous output line was interrupted (ended with **\c**), 0 otherwise.

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

\n[.lg] Interpolate ligature mode.

\n[.linetabs] Interpolate 1 if line-tabs mode is enabled, 0 otherwise.

\n[.ll] Interpolate line length applicable to the pending output line.

\n[.lt] Interpolate title line length.

\n[.m] Interpolate name of the selected stroke color. This is a string-valued register.

\n[.M] Interpolate name of the selected fill color. This is a string-valued register.

\n[.ne] Interpolate amount of space demanded by the most recent **ne** request that caused a page location trap to be sprung. See **\n[.trunc]**.

\n[.nm] Interpolate 1 if output line numbering is enabled (even if temporarily suppressed), 0 otherwise.

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

\n[.O] Interpolate output suppression level. See **\O**.

\n[.P] Interpolate 1 if the current page is selected for output. See **−o** command-line option to *troff* (1).

\n[.pe] Interpolate 1 during page ejection, 0 otherwise.

\n[.pn] Interpolate next page number (either that set by **pn**, or that of the current page plus 1).

\n[.ps] Interpolate type size in scaled points.

\n[.psr] Interpolate most recently requested type size in scaled points.

\n[.pvs] Interpolate post-vertical line spacing amount.

\n[.rj] Interpolate number of input lines remaining to be right-aligned.

\n[.slant] Interpolate font slant. See **\S**.

\n[.sr] Interpolate most recently requested type size in points as a decimal fraction. This is a string-valued register.

\n[.ss]
\n[.sss] Interpolate values of minimal inter-word space and additional inter-sentence space, respectively, in twelfths of the space width of the selected font.

\n[.sty] Interpolate selected abstract font style, if any. This is a string-valued register.

\n[.tabs] Interpolate representation of the tab stop settings in a form suitable for passage to the **ta** request.

\n[.trunc] Interpolate amount of vertical space truncated by the most recently sprung page location 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, **\n[.trunc]** represents the difference of what the vertical position would have been but for the trap, and what the vertical position actually is. See **\n[.ne]**.

\n[.U] Interpolate 1 if in unsafe mode, 0 otherwise. See **−U** command-line option to *troff* (1).

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

\n[.warn] Interpolate warning mode. See section "Warnings" of *troff* (1).

\n[.x] Interpolate major version number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.x]** contains 1.

\n[.y] Interpolate minor version number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.y]** contains 23.

\n[.Y] Interpolate revision number of the running *troff* formatter. For example, if the version number is 1.23.0, then **\n[.Y]** contains 0.

\n[.zoom] Interpolate magnification of font, in thousandths, or 0 if magnification unused. See **fzoom**.

The following (writable) registers are set by the **psbb** request.

\n[llx]
\n[lly]
\n[urx]
\n[ury] Interpolate the (upper, lower, left, right) bounding box values (in PostScript units) of the most recently processed PostScript image.

The following (writable) registers are set by the **\w** escape sequence.

\n[rst]
\n[rsb] Like **\n[st]** and **\n[sb]**, but taking account of the heights and depths of glyphs. In other words, these registers store the highest and lowest vertical positions attained by the argument formatted by the **\w** escape sequence, doing what AT&T *troff* documented **\n[st]** and **\n[sb]** as doing.

\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 writable registers are as follows. Those relating to date and time are initialized using *localtime*(3) at formatter startup.

\n[c.] Interpolate input line number. **\n[.c]** is a read-only alias of this register.

\n[hours] Interpolate number of hours elapsed since midnight.

\n[hp] Interpolate horizontal position relative to that at the start of the input line.

\n[lsn]
\n[lss] Interpolate count of leading spaces on input line and amount of corresponding horizontal motion, respectively.

\n[minutes] Interpolate number of minutes elapsed in the hour.

\n[seconds] Interpolate number of seconds elapsed in the minute.

\n[systat] Interpolate return value of *system*(3) function executed by most recent **sy** request.

\n[slimit] Interpolates maximum quantity of objects on *troff*'s internal input stack (default: 1000). If non-positive, there is no limit: recursion can continue until program memory is exhausted.

\n[year] Interpolate Gregorian year. AT&T *troff*'s **\[yr]** interpolates the Gregorian year minus 1900.

Miscellaneous

GNU *troff* predefines one string, **.T**, containing the argument given to the **−T** command-line option, namely the output device (for example, **pdf** or **utf8**). The (read-only) *register* **.T** interpolates 1 if GNU *troff* is run with the **−T** command-line option, and 0 otherwise.

A font not listed in the output device's *DESC* file's **fonts** directive is automatically mounted at the next available font position when it is selected. If you mount a font explicitly with the **fp** request, you should do so on the first unused position, which can be found in the **.fp** register.

Unparameterized string interpolation does not conceal the arguments to a macro being interpreted. Thus, in a macro definition, the call of another macro with the existing argument list,

```
.xx \\$@
```

is more efficiently done with

```
\\*[xx]\\
```

(that is, with string interpolation). The trailing backslashes prevent the final newline in the macro definition from being interpolated, potentially putting an unwanted blank line on the output. See section "Punning Names" in *groff*(7).

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

GNU *troff* keeps track of the nesting depth of escape sequence interpolations and other uses of delimiters, as in the **tl** request and the output comparison operator (that is, input like **'foo'bar'** as a conditional expression), so the only characters you need to avoid using as delimiters are those that appear in the arguments you input, not any that result from interpolation. Typically, **'** works fine. Use visible characters as delimiters in GNU *troff*, not "ASCII" controls like BEL (Control+G). The implementation of **\$@** ensures that the double quotes surrounding an argument appear at an interpolation depth different from that of the arguments themselves. Similarly, in bracket-form escape sequences like **\f[ZCMI],** a right bracket **]** does not end the sequence unless it occurs at the same interpolation depth as the opening **[**, so input like

```
\f[\*[my-family]\*[my-style]]
```

works as desired. In compatibility mode, no attention is paid to the interpolation depth.

In GNU *troff*, the **tr** request can map characters to the unbreakable space escape sequence **\~** as a special case (**tr** normally operates only on *characters*). This feature replaces the odd-parity **tr** mapping trick used in AT&T *troff* documents, where a character, often **~**, was "sacrificed" by mapping it to "nothing", drafting it into use as an unadjustable, unbreakable space. (This feature was gratuitous even in early AT&T *troff,* which supported the *****space* escape sequence by 1976.) Often, it makes more sense to use GNU *troff*'s **\~** escape sequence instead, which has been adopted by every other active *troff* implementation except that of Illumos, as well as by the non-*troff* *mandoc*. Translation of a character to **\~** is unnecessary.

GNU *troff* permits tabs and spaces after the first dot on a control line that ends a macro definition.

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

Formatter output

The page description language output by GNU *troff* is modeled after that used by AT&T *troff* once the latter adopted a device-independent approach in the early 1980s. Only the differences are documented here. For a fuller discussion, see *groff_out*(5).

Glyph and font names can be of arbitrary length; postprocessors should not assume that they are at most two characters. A glyph to be formatted is always drawn from the current font; in contrast to AT&T device-independent *troff* , drivers need not search special fonts to find a glyph.

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 H**" command is also in scaled points.

Simple commands

If the **tcommand** directive is present in the output device's *DESC* file, GNU *troff* employs the following two commands.

t *xyz*...
: Typeset word *xyz*; that is, set a sequence of ordinary glyphs named *x*, *y*, *z*, ..., terminated by a space or newline; an optional second integer argument is ignored (this allows the formatter to generate an even number of arguments). Each glyph is set at the current drawing position, and the position is then advanced horizontally by the glyph's width. A glyph's width is read from its metrics in the font description file, scaled to the current type size, and rounded to a multiple of the horizontal motion quantum. Use the **C** command to emplace glyphs of special characters.

u *n xyz*...
: Typeset word *xyz* with track kerning. As **t**, but after placing each glyph, the drawing position is further advanced horizontally by *n* basic units.

New commands implement color support.

mc *cyan magenta yellow*
md
mg *gray*
mk *cyan magenta yellow black*
mr *red green blue*
: Set the components of the stroke color with respect to various color spaces. **md** resets the stroke color to the default value. The arguments are integers in the range 0 to 65535.

A new device control subcommand is available.

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

Extended drawing commands

GNU *pic* does not produce *troff* escape sequences employing these extensions if its **−n** option is given.

Df *n*
: Set the shade of gray used to fill geometric objects to *n*, which must be an integer. 0 corresponds to white and 1000 to black. A grayscale ramp spans the two. A value outside this range uses the stroke color as the fill color. The fill color is opaque. Normally the default is black, but some drivers may provide a way of changing this. **Df** is obsolete since 2002, superseded by **DFg** below.

  The corresponding **\D'f'** escape sequence should not be used: its argument is rounded to an integer multiple of the horizontal motion quantum, which can limit the precision of *n*.

DC *d*
: Draw a filled circle of diameter *d* with its leftmost point at the drawing position.

DE *h v*
: Draw a filled ellipse, of horizontal axis *h* and vertical axis *v*, with its leftmost point at the drawing position.
  delim $$

Dp $dx sub 1 ~ dy sub 1 ~ ldots ~ dx sub n ~ dy sub n$
: Draw a polygon with, for $i = 1 , ldots , n + 1$, its *i*th vertex at the drawing position $+ sum from { j = 1 } to { i - 1 } (dx sub j , dy sub j)$. *groff* output drivers automatically close polygons,

drawing a line from $(dx sub n , dy sub n)$ back to $(dx sub 1 , dy sub 1)$. The drawing position is left at the last *specified* vertex, but this may change in a future version of GNU *troff*. Heirloom Doctools *troff*, like DWB *troff*, by default does not close the polygon. In its *groff* compatibility mode, Heirloom closes the polygon but leaves the drawing position *unchanged*—that is, at the polygon's *initial* drawing position.

At the moment, GNU *pic* uses this command only to generate triangles and rectangles.

DP $dx sub 1 ~ dy sub 1 ~ ldots ~ dx sub n ~ dy sub n$
As **Dp**, but draw a filled rather than a stroked polygon.

Dt *n* Set the line thickness to *n* basic units. AT&T *troff* output drivers use a thickness proportional to the type size; this is the GNU *troff* default. A negative *n* requests this explicitly. An *n* of zero selects the smallest available line thickness.

A difficulty arises in how the drawing position should be changed after the execution of these commands. This has little importance to most users, since the output of GNU *grn* and *pic* does not depend on it. Given a drawing command of the form **D***z* $x sub 1 ~ y sub 1 ~ ldots ~ x sub n ~ y sub n$, where *z* is not **c** or **e**, AT&T *troff* treats each $x sub i$ as a horizontal motion, each $y sub i$ as a vertical one, and therefore assumes that the width of the drawn object is $sum from { i = 1 } to n x sub i$, and its height is $sum from { i = 1 } to n y sub i$. (Verify its assumption about height by examining the **st** and **sb** registers after using such a drawing command in a **\w** escape sequence). For the sake of compatibility, GNU *troff* also follows this rule, even though it frustrates extensions to the **D** command that set drawing parameters rather than rendering objects, producing ugly results in the case of **Dt** and **Df**, or otherwise don't parameterize objects as a series of vertices, as with GNU *troff*'s filled ellipse, **DE**. Thus after executing a **D** command of the form **D***z* $x sub 1 ~ y sub 1 ~ ldots ~ x sub n ~ y sub n$, the drawing position should be increased by $(sum from { i = 1 } to n x sub i , sum from { i = 1 } to n y sub i)$. delim off In a future release, GNU *troff* and its output drivers may abandon the application of this assumption to drawing commands not explicitly specified in the AT&T "Troff User's Manual".

Fill color selection is implemented with another set of extensions.

DFc *cyan magenta yellow*
DFd
DFg *gray*
DFk *cyan magenta yellow black*
DFr *red green blue*
Set the components of the fill color as described under the **\M** escape sequence above. **DFd** restores the device's default fill color. The drawing position is not updated, in contrast to **Df**.

Device control syntax extension

GNU *troff* introduces a line continuation convention, permitting the argument to the **x X** command to contain newlines. A newline in the input is transformed to the sequence "*newline*+". When interpreting an **x X** command, a postprocessor should therefore be prepared for a plus sign after a newline; if it occurs, preserve the newline, discard the plus sign, and continue to collect the input into the argument of the **x X** command. A newline *not* followed by a plus sign terminates the **x X** command. An application of this feature is the embedding of PostScript or PDF language command streams into *troff* output.

GNU *troff* guarantees that the first three output commands it emits are as follows.

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

Debugging

In addition to AT&T *troff*'s debugging features, GNU *troff* emits more error diagnostics when syntactical or semantic nonsense is encountered and supports several warning categories; the output of these can be selected with **warn**. Also see the **−E**, **−w**, and **−W** options of *troff*(1). Backtraces can be automatically produced when errors or warnings occur (the **−b** option of *troff*(1)) or generated on demand (**backtrace**).

groff also adds more flexible diagnostic output requests (**tmc** and **tm1**). More aspects of formatter state

can be examined with requests that write lists of defined registers (**pnr**), environments (**pev**), and page location traps (**ptr**) to the standard error stream.

Implementation differences

GNU *troff*'s features sometimes cause incompatibilities with documents written assuming old implementations of *troff*. Some GNU extensions to *troff* are supported by other implementations.

When adjusting to both margins, AT&T *troff* at first adjusts spaces starting from the right; GNU *troff* begins from the left. Both implementations adjust spaces from opposite ends on alternating output lines to prevent "rivers" in the text.

GNU *troff* does not always hyphenate words as AT&T *troff* does. The AT&T implementation uses a set of hard-coded rules specific to U.S. English, while GNU *troff* uses language-specific hyphenation pattern files derived from TEX. In some versions of *troff* there was limited space to store hyphenation exceptions (arguments to the **hw** request); GNU *troff* has no such restriction.

Long names may be GNU *troff*'s most obvious innovation. AT&T *troff* interprets "**.dsabcd**" as defining a string "**ab**" with contents "**cd**". Normally, GNU *troff* interprets this as a call of a macro named "**dsabcd**". AT&T *troff* also interprets ***[** and **\n[** as an interpolation of a string or register, respectively, called "**[**". In GNU *troff*, however, the "**[**" is normally interpreted as beginning the enclosure of a long identifier. In compatibility mode, GNU *troff* interprets names in the traditional way, which means that they are limited to one or two characters. See the **−C** option in *troff*(1) and, above, the **.C** and **.cp** registers, and **cp** and "**do**" requests, for more on compatibility mode.

The register **\n[.cp]** is specialized and may require a statement of rationale. When writing macro packages or documents that use GNU *troff* features and which may be mixed with other packages or documents that do not—common scenarios include serial processing of man pages or use of the "**so**" or **mso** requests—you may desire correct operation regardless of compatibility mode enablement in the surrounding context. It may occur to you to save the existing value of **\n(.C** into a register, say, **_C**, at the beginning of your file, turn compatibility mode off with "**.cp 0**", then restore it from that register at the end with "**.cp \n(_C**". At the same time, a modular design of a document or macro package may lead you to multiple layers of inclusion. You cannot use the same register name everywhere lest you "clobber" the value from a preceding or enclosing context. The two-character register name space of AT&T *troff* is confining and mnemonically challenging; you may wish to use GNU *troff*'s more capacious name space. However, attempting "**.nr _my_saved_C \n(.C**" will not work in compatibility mode; the register name is too long. "This is exactly what **.do** is for," you think, "**.do nr _my_saved_C \n(.C**". The foregoing will always save zero to your register, because "**do**" turns compatibility mode *off* while it interprets its argument list. What you need is:

```
.do nr _my_saved_C \n[.cp]
.cp 0
```

at the beginning of your file, followed by

```
.cp \n[_my_saved_C]
.do rr _my_saved_C
```

at the end. As in the C language, we all have to share one big name space, so choose a register name that is unlikely to collide with other uses.

The existence of the **.T** string is a common feature of post-CSTR #54 *troff*s—DWB 3.3, Solaris, Heirloom Doctools, and Plan 9 *troff* all support it—but valid values are specific to each implementation. The behavior of the **.T** register in GNU *troff* differs from AT&T *troff*, which interpolated 1 only if *nroff* was the formatter and was called with **−T**.

The **lf** request sets the number of the *current* input line in AT&T *troff*, and the *next* in GNU *troff*.

AT&T *troff* had only environments named "**0**", "**1**", and "**2**". In GNU *troff*, any number of environments may exist, using any valid identifiers for their names.

GNU *troff* normally tracks the interpolation depth of escape sequence parameters and other delimited structures, but not in compatibility mode. See section "Miscellaneous" above.

In compatibility mode, the escape sequences **\f**, **\H**, **\m**, **\M**, **\R**, **\s**, and **\S** are transparent at the beginning of an input line for the purpose of recognizing a control character, because they modify formatter state (**\R**)

or properties of the environment (the rest) and therefore do not create output nodes. For example, this code produces bold output in both cases, but the text differs,

```
.de xx '
Hello!
..
\fB.xx\fP
```

formatting ".xx" normally and "Hello!" in compatibility mode.

GNU *troff* request names unrecognized by other *troff* implementations will likely be ignored; escape sequences that are GNU *troff* extensions are liable to format their function selector character. For example, the adjustable, non-breaking space escape sequence **\~** is also supported by Heirloom Doctools *troff* 050915 (September 2005), *mandoc* 1.9.5 (2009-09-21), *neatroff* (commit 1c6ab0f6e, 2016-09-13), and Plan 9 from User Space *troff* (commit 93f8143600, 2022-08-12), but not by Solaris/Illumos *troff*s, which will render it as **~**.

GNU *troff* does not allow the use of the escape sequences **\|**, **\^**, **\&**, **\{**, **\}**, *****space*, **\'**, **\`**, **\–**, **_**, **\!**, **\%**, or **\c** in identifiers; AT&T *troff* does. The **\A** escape sequence (see subsection "Escape sequences" above) may be helpful in avoiding their use.

Normally, the syntax form **\s***n* accepts only a single character (a digit) for *n*, consistently with other forms that originated in AT&T *troff*, like *****, **\$**, **\f**, **\g**, **\k**, **\n**, and **\z**. In compatibility mode only, a non-zero *n* must be in the range 4–39. Legacy documents relying upon this quirk of parsing should be migrated to another **\s** form. [Background: The Graphic Systems C/A/T phototypesetter (the original device target for AT&T *troff*) supported only a few discrete type sizes in the range 6–36 points, so Ossanna contrived a special case in the parser to do what the user must have meant. Kernighan warned of this in the 1992 revision of CSTR #54 (§2.3), and more recently, McIlroy referred to it as a "living fossil".]

Fractional type sizes cause one noteworthy incompatibility. In AT&T *troff* the **ps** request ignores scaling units and thus "**.ps 10u**" sets the type size to 10 points, whereas in GNU *troff* it sets the type size to 10 *scaled* points, which may be a much smaller measurement. See subsection "Fractional type sizes and new scaling units" above.

The **ab** request differs from AT&T *troff*: GNU *troff* writes no message to the standard error stream if no arguments are given, and it exits with a failure status instead of a successful one.

The **bp** request differs from AT&T *troff*: GNU *troff* does not accept a scaling unit on the argument, a page number; the former (somewhat uselessly) does.

In AT&T *troff* the **pm** request reports macro, string, and diversion sizes in units of 128-byte blocks, and an argument reduces the report to a sum of the above in the same units. GNU *troff* ignores any arguments and reports the sizes in bytes.

Unlike AT&T *troff*, GNU *troff* does not ignore the **ss** request if the output is a terminal device; instead, the values of minimum inter-word and additional inter-sentence space are each rounded down to the nearest multiple of 12.

In GNU *troff* there is a fundamental difference between (unformatted) characters and (formatted) glyphs. Everything that affects how a glyph is output is stored with the glyph node; once a glyph node has been constructed, it is unaffected by any subsequent requests that are executed, including **bd**, **cs**, **tkf**, **tr**, or **fp** requests. Normally, glyphs are constructed from characters immediately before the glyph is added to an output line. Macros, diversions, and strings are all, in fact, the same type of object; they contain a sequence of intermixed character and glyph nodes. Special characters transform from one to the other: before being added to the output, they behave as characters; afterward, they are glyphs. A glyph node does not behave like a character node when it is processed by a macro: it does not inherit any of the special properties that the character from which it was constructed might have had. For example, the input

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

produces "\\" in GNU *troff*. Each pair of backslashes becomes one backslash *glyph;* the resulting backslashes are thus not interpreted as escape *characters* when they are reread as the diversion is output. AT&T *troff would* interpret them as escape characters when rereading them and end up printing one "\".

One way to format a backslash in most documents is with the **\e** escape sequence; this formats the glyph of the current escape character, regardless of whether it is used in a diversion; it also works in both GNU *troff* and AT&T *troff*. (Naturally, if you've changed the escape character, you need to prefix the "**e**" with whatever it is—and you'll likely get something other than a backslash in the output.)

The other correct way, appropriate in contexts independent of the backslash's common use as a *roff* escape character—perhaps in discussion of character sets or other programming languages—is the character escape **\(rs** or **\[rs]**, for "reverse solidus", from its name in the ECMA-6 (ISO/IEC 646) standard. [This escape sequence is not portable to AT&T *troff*, but is to its lineal descendant, Heirloom Doctools *troff*, as of its 060716 release (July 2006).]

To store an escape sequence in a diversion that is interpreted when the diversion is reread, either use the traditional **\!** transparent output facility, or, if this is unsuitable, the new **\?** escape sequence. See subsection "Escape sequences" above and sections "Diversions" and "gtroff Internals" in *Groff: The GNU Implementation of troff*, the *groff* Texinfo manual.

In the somewhat pathological case where a diversion exists containing a partially collected line and a partially collected line at the top-level diversion has never existed, AT&T *troff* will output the partially collected line at the end of input; GNU *troff* will not.

Formatter output incompatibilities

Its extensions notwithstanding, the *groff* intermediate output format has some incompatibilities with that of AT&T *troff*, but better compatibility is sought; problem reports and patches are welcome. The following incompatibilities are known.

- The drawing position after rendering polygons is inconsistent with AT&T *troff* practice. Other implementations have diverged on this point as well.
- The output cannot be easily rescaled to other devices as AT&T *troff*'s could.

Authors

This document was written by James Clark ⟨jjc@jclark.com⟩, Werner Lemberg ⟨wl@gnu.org⟩, Bernd Warken ⟨groff−bernd.warken−72@web.de⟩, and G. Branden Robinson ⟨g.branden.robinson@gmail.com⟩.

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".

"Troff User's Manual" by Joseph F. Ossanna, 1976 (revised by Brian W. Kernighan, 1992), AT&T Bell Laboratories Computing Science Technical Report No. 54, widely called simply "CSTR #54", documents the language, device and font description file formats, and output format referred to collectively in *groff* documentation as AT&T *troff*.

"A Typesetter-independent TROFF" by Brian W. Kernighan, 1982, AT&T Bell Laboratories Computing Science Technical Report No. 97, provides additional insights into the device and font description file formats and output format.

groff (1), *groff* (7), *roff* (7)