Name

eqn − format equations for *groff* or MathML

Synopsis

eqn [**−CNrR**] [−d *xy*] [**−f** *F*] [**−m** *n*] [**−M** *dir*] [**−p** *n*] [**−s** *n*] [**−T** *name*] [*file* ...]

eqn −−help

eqn −v
eqn −−version

Description

The GNU implementation of *eqn* is part of the *groff* (7) document formatting system. *eqn* is a *tr off* (1) preprocessor that translates descriptions of equations embedded in *roff* (7) input files into the language understood by *troff* (1). It copies the contents of each *file* to the standard output stream, e xcept that lines between **.EQ** and **.EN** (or within a pair of user-specified delimiters) are interpreted as equation descriptions. Normally, *eqn* is not executed directly by the user, but invoked by specifying the **−e** option to *groff* (1). While GNU *eqn*'s input syntax is highly compatible with AT&T *eqn*, the output *eqn* produces cannot be processed by AT&T *troff*; GNU *troff* (or a *troff* implementing relevant GNU extensions) must be used. If no *file* operands are given on the command line, or if *file* is "−", *eqn* reads the standard input stream.

Unless the **−R** option is given, *eqn* searches for the file *eqnrc* in the directories given with the **−M** option first, then in */usr/local/share/groff/site−tmac*, and finally in the standard macro directory */usr/local/share/groff/1.23.0/tmac*. If it exists and is readable, *eqn* processes it before any input files.

This man page primarily discusses the differences between GNU *eqn* and AT&T *eqn*. Most of the new features of the GNU *eqn* input language are based on TEX. There are some references to the differences between TEX and GNU *eqn* below; these may safely be ignored if you do not know TEX.

Three points are worth special note.

- GNU *eqn* emits Presentation MathML output when invoked with the "**−T MathML**" option.
- GNU *eqn* does not provide the functionality of *neqn*: it does not support low-resolution, typewriter-like devices (although it may work adequately for very simple input).
- GNU *eqn* sets the input token "**...**" as three periods or low dots, rather than the three centered dots of AT&T *eqn*. To get three centered dots, write **cdots** or "**cdot cdot cdot**".

Anatomy of an equation

eqn input consists of tokens. Consider a form of Newton's second law of motion. The input

```
.EQ
F =
m a
.EN
```

becomes $F = ma$. Each of **F**, **=**, **m**, and **a** is a tok en. Spaces and newlines are interchangeable; they separate input tokens but do not break lines or produce space in the output.

A handful of tokens manage the grouping and spacing of other tokens.

{ } Braces perform grouping. Whereas "**e sup a b**" expresses $e^a b$, "**e sup { a b }**" meanse ^{ab}.

^

~ are the *half space* and *full space*, respectively. Use them to tune the appearance of the output.

Tab and leader characters separate tokens as well as advancing the drawing position to the next tab stop, but are seldom used in *eqn* input. When they occur, they must appear at the outermost lexical scope. This roughly means that they can't appear within braces that are necessary to disambiguate the input; *eqn* will diagnose an error in this event.

Other tokens are primitives, macros, an argument to either of the foregoing, or a component of an equation.

Primitives are fundamental keywords of the *eqn* language. They can configure an aspect of the preprocessor's state, as when setting a "global" font selection or type size (**gfont** and **gsize**), or declaring or deleting

macros (**define** and **undefine**); these are termed *commands.* Other primitives perform formatting operations on the tokens after them (as with **fat**, **over**, **sqrt**, or **up**).

Macros permit primitives and/or components of equations to be grouped together for convenience. Predefined macros make convenient the preparation of *eqn* input in a form resembling its spoken expression; for example, consider **cos**, **hat**, **inf**, and **lim**.

Equation *components* include mathematical variables, constants, numeric literals, and operators. *eqn* remaps some input character sequences for convenience in equation entry and to ensure that glyphs from an unstyled font are used; see *groff_char*(7).

```
+    \[pl]
-    \[mi]
=    \[eq]
'    \[fm]
<=   \[<=]
>=   \[>=]
```

Spacing and typeface

GNU *eqn* imputes types to the components of an equation, adjusting the spacing between them accordingly. Recognized types are as follows; most affect spacing only, whereas the "**letter**" subtype of "**ordinary**" also assigns a style.

`ordinary`	character such as "1", "a", or "!"
`letter`	character to be italicized by default
`digit`	*n/a*
`operator`	large operator such as " $\sum$"
`binary`	binary operator such as "+"
`relation`	relation such as "="
`opening`	opening bracket such as "("
`closing`	closing bracket such as ")"
`punctuation`	punctuation character such as ","
`inner`	sub-formula contained within brackets
`suppress`	component to which automatic spacing is not applied

Two primitives apply types to equation components.

type *t e* Apply type *t* to expression *e*.

chartype *t text*
: Assign each character in (unquoted) *text* type *t*, persistently.

eqn sets up spacings and styles as if by the following commands.

```
chartype "letter"      abcdefghiklmnopqrstuvwxyz
chartype "letter"      ABCDEFGHIKLMNOPQRSTUVWXYZ
chartype "letter"      \[*a]\[*b]\[*x]\[*d]\[*e]\[*y]
chartype "letter"      \[*g]\[*i]\[*k]\[*l]\[*m]\[*n]
chartype "letter"      \[*w]\[*o]\[*f]\[*p]\[*q]\[*r]
chartype "letter"      \[*s]\[*t]\[*h]\[*u]\[*c]\[*z]
chartype "binary"      *\[pl]\[mi]
chartype "relation"    <>\[eq]\[<=]\[>=]
chartype "opening"     {([
chartype "closing"     })]
chartype "punctuation" ,;:.
chartype "suppress"    ^~
```

eqn assigns all other ordinary and special *roff* characters, including numerals 0–9, the "**ordinary**" type. (The "**digit**" type is not used, but is available for customization.) In keeping with common practice in mathematical typesetting, lowercase, but not uppercase, Greek letters are assigned the "**letter**" type to style them in italics.

Primitives

eqn supports without alteration the AT&T *eqn* primitives **above**, **back**, **bar**, **bold**, **define**, **down**, **fat**, **font**, **from**, **fwd**, **gfont**, **gsize**, **italic**, **left**, **lineup**, **mark**, **matrix**, **ndefine**, **over**, **right**, **roman**, **size**, **sqrt**, **sub**, **sup**, **tdefine**, **to**, **under**, and **up**.

New primitives

The GNU extension primitives "**type**" and**chartype** are discussed in subsection "Spacing and typef ace" above; "**set**" in subsection "Customization" below; and **grfont** and **gbfont** in subsection "Fonts" below.

big *e* Enlarges the expression *e*; intended to have semantics like CSS "large". In*troff* output, the type size is increased by 5. MathML output emits the following.

```
<mstyle mathsize='big'>
```

e1 **smallover** *e2*

This is similar to **over**; **smallover** reduces the size of *e1* and *e2*; it also puts less vertical space between *e1* or *e2* and the fraction bar. The**over** primitive corresponds to the TEX **\over** primitive in displayed equation styles; **smallover** corresponds to **\over** in non-display ("inline") styles.

vcenter *e*

This vertically centers *e* about the math axis. The math axis is the vertical position about which characters such as "+" and "−" are centered; it is also the vertical position used for fraction bars. For example, **sum** is defined as follows.

```
{ type "operator" vcenter size +5 \(*S }
```

vcenter is silently ignored when generating MathML.

e1 **accent** *e2*

This sets *e2* as an accent over *e1*. *e2* is assumed to be at the correct height for a lowercase letter; *e2* is moved down according to whether *e1* is taller or shorter than a lowercase letter. For example, **hat** is defined as follows.

```
accent { "^" }
```

dotdot, **dot**, **tilde**, **vec**, and **dyad** are also defined using the **accent** primitive.

e1 **uaccent** *e2*

This sets *e2* as an accent under *e1*. *e2* is assumed to be at the correct height for a character without a descender; *e2* is moved down if *e1* has a descender. **utilde** is predefined using **uaccent** as a tilde accent below the baseline.

split "*text*"

This has the same effect as simply

```
text
```

but *text* is not subject to macro expansion because it is quoted; *text* is split up and the spacing between individual characters adjusted per subsection "Spacing and typeface" above.

nosplit *text*

This has the same effect as

```
"text"
```

but because *text* is not quoted it is subject to macro expansion; *text* is not split up and the spacing between individual characters *not* adjusted per subsection "Spacing and typeface" above.

e **opprime**

This is a variant of **prime** that acts as an operator on *e*. It produces a different result from **prime** in a case such as "**A opprime sub 1**": with **opprime** the "1" is tucked under the prime as a subscript to the "A" (as is conventional in mathematical typesetting), whereas with **prime** the "1" is a subscript to the prime character. The precedence of**opprime** is the same as that of **bar** and **under**, which is higher than that of everything except **accent** and **uaccent**. In unquoted text, a neutral apostrophe (') that is not the first character on the input line is treated like **opprime**.

special *troff-macro e*

Construct an object by calling *troff-macro* on *e*. The*troff* string **0s** contains the *eqn* output for *e*, and the registers **0w**, **0h**, **0d**, **0skern**, and **0skew** the width, height, depth, subscript kern, and skew of *e*, respectively. (The*subscript kern* of an object indicates how much a subscript on that object should be "tucked in", or placed to the left relative to a non-subscripted glyph of the same size. The *skew* of an object is how far to the right of the center of the object an accent over it should be placed.) The macro must modify**0s** so that it outputs the desired result, returns the drawing position to the text baseline at the beginning of *e*, and updates the foregoing registers to correspond to the new dimensions of the result.

Suppose you want a construct that "cancels" an expression by drawing a diagonal line through it.

```
.de Ca
.  ds 0s \
\Z'\\*(0s'\
\v'\\n(0du'\
\D'l \\n(0wu −\\n(0hu−\\n(0du'\
\v'\\n(0hu'
..
.EQ
special Ca "x \[mi] 3 \[pl] x" ~ 3
.EN
```

We use the **\[mi]** and **\[pl]** special characters instead of + and − because they are part of the argument to a *troff* macro, so *eqn* does not transform them to mathematical glyphs for us. Here's a more complicated construct that draws a box around an expression; the bottom of the box rests on the text baseline. We define the *eqn* macro **box** to wrap the call of the *troff* macro **Bx**.

```
.de Bx
.ds 0s \
\Z'\\h'1n'\\*[0s]'\
\v'\\n(0du+1n'\
\D'l \\n(0wu+2n 0'\
\D'l 0 −\\n(0hu−\\n(0du−2n'\
\D'l −\\n(0wu−2n 0'\
\D'l 0 \\n(0hu+\\n(0du+2n'\
\h'\\n(0wu+2n'
.nr 0w +2n
.nr 0d +1n
.nr 0h +1n
..
.EQ
define box ' special Bx $1 '
box(foo) ~ "bar"
.EN
```

space *n*

Set extra vertical spacing around the equation, replacing the default values, where *n* is an integer in hundredths of an em. If positive, *n* increases vertical spacing before the equation; if negative, it does so after the equation. This primitive provides an interface to *groff*'s **\x** escape sequence, but with the opposite sign convention. It has no effect if the equation is part of a *pic*(1) picture.

Extended primitives

eqn recognizes an "**on**" argument to the **delim** primitive specially, restoring any delimiters previously disabled with "**delim off**". If delimiters haven't been specified, neither command has effect.

col *n* **{** . . . **}**
ccol *n* **{** . . . **}**
lcol *n* **{** . . . **}**
rcol *n* **{** . . . **}**
pile *n* **{** . . . **}**
cpile *n* **{** . . . **}**
lpile *n* **{** . . . **}**
rpile *n* **{** . . . **}**

The integer value *n* (in hundredths of an em) increases the vertical spacing between rows, using *groff*'s **\x** escape sequence (the value has no effect in MathML mode). Negative values are accepted but have no effect. If more than one *n* occurs in a matrix, the lar gest is used.

Customization

When *eqn* generates *troff* input, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML, which delegates typesetting to a MathML rendering engine. Configure these parameters with the **set** primitive.

set *p n* assigns parameter *p* the inte ger value *n*; *n* is interpreted in units of hundredths of an em unless otherwise stated. For example,

```
set x_height 45
```

says that *eqn* should assume that the font's x-height is 0.45 ems.

Available parameters are as follows; defaults are shown in parentheses. We intend these descriptions to be expository rather than rigorous.

minimum_size sets a floor for the type size (in scaled points) at which equations are set (**5**).

fat_offset
The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount (**4**). **fat_offset** is not used in MathML mode; **fat** components use

```
<mstyle mathvariant='double-struck'>
```

instead.

over_hang A fraction bar is longer by twice this amount than the maximum of the widths of the numerator and denominator; in other words, it overhangs the numerator and denominator by at least this amount (**0**).

accent_width When **bar** or **under** is applied to a single character, the line is this long (**31**). Normally, **bar** or **under** produces a line whose length is the width of the object to which it applies; in the case of a single character, this tends to produce a line that looks too long.

delimiter_factor Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth of at least this many thousandths of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis (**900**).

delimiter_shortfall Extensible delimiters produced with the **left** and **right** primitives have a combined height and depth not less than the difference of twice the maximum amount by which the sub-equation that the delimiters enclose extends away from the axis and this amount (**50**).

null_delimiter_space
This much horizontal space is inserted on each side of a fraction (**12**).

script_space The width of subscripts and superscripts is increased by this amount (**5**).

thin_space This amount of space is automatically inserted after punctuation characters. It also configures the width of the space produced by the ^ token (**17**).

medium_space This amount of space is automatically inserted on either side of binary operators (**22**).

thick_space This amount of space is automatically inserted on either side of relations. It also configures the width of the space produced by the ~ token (**28**).

x_height The height of lowercase letters without ascenders such as "x" (**45**).

axis_height The height above the baseline of the center of characters such as "+" and "−" (**26**). It is important that this value is correct for the font you are using.

default_rule_thickness
This should be set to the thickness of the **\[ru]** character, or the thickness of horizontal lines produced with the **\D** escape sequence (**4**).

num1 The **over** primitive shifts up the numerator by at least this amount (**70**).

num2 The **smallover** primitive shifts up the numerator by at least this amount (**36**).

denom1 The **over** primitive shifts down the denominator by at least this amount (**70**).

denom2 The **smallover** primitive shifts down the denominator by at least this amount (**36**).

sup1 Normally superscripts are shifted up by at least this amount (**42**).

sup2 Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount (**37**). Conventionally, this is less than **sup1**.

sup3 Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount (**28**). Conventionally, this is less than **sup2**.

sub1 Subscripts are normally shifted down by at least this amount (**20**).

sub2 When there is both a subscript and a superscript, the subscript is shifted down by at least this amount (**23**).

sup_drop The baseline of a superscript is no more than this much below the top of the object on which the superscript is set (**38**).

sub_drop The baseline of a subscript is at least this much below the bottom of the object on which the subscript is set (**5**).

big_op_spacing1 The baseline of an upper limit is at least this much above the top of the object on which the limit is set (**11**).

big_op_spacing2 The baseline of a lower limit is at least this much below the bottom of the object on which the limit is set (**17**).

big_op_spacing3 The bottom of an upper limit is at least this much above the top of the object on which the limit is set (**20**).

big_op_spacing4 The top of a lower limit is at least this much below the bottom of the object on which the limit is set (**60**).

big_op_spacing5 This much vertical space is added above and below limits (**10**).

baseline_sep The baselines of the rows in a pile or matrix are normally this far apart (**140**). Usually equal to the sum of**num1** and **denom1**.

shift_down The midpoint between the top baseline and the bottom baseline in a matrix or pile is shifted down by this much from the axis (**26**). Usually equal to **axis_height**.

column_sep This much space is added between columns in a matrix (**100**).

matrix_side_sep This much space is added at each side of a matrix (**17**).

draw_lines If non-zero, *eqn* draws lines using the *troff* **\D** escape sequence, rather than the **\l** escape sequence and the **\[ru]** special character. The default is determined by the *eqnrc* file (**0** on most devices; **1** on **ps**, **html**, and the X11 devices.)

body_height The amount by which the height of the equation exceeds this is added as extra space before the line containing the equation using the *troff* **\x** escape sequence (**85**).

body_depth The amount by which the depth of the equation exceeds this is added as extra space after the line containing the equation using the *troff* **\x** escape sequence (**35**).

nroff If non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The default is determined by the *eqnrc* file (**0** on most devices; **1** on **ascii**, **latin1**, **utf8**, and **cp1047**).

Macros

In GNU *eqn*, macros can take arguments. In a macro body, **$***n*, where *n* is between 1 and 9, is replaced by the *n*th argument if the macro is called with arguments; if there are fewer than *n* arguments, it is replaced by nothing. A word containing a left parenthesis where the part of the word before the left parenthesis has been defined using the **define** primitive is recognized as a macro call with arguments; characters following the left parenthesis up to a matching right parenthesis are treated as comma-separated arguments. Commas inside nested parentheses do not terminate an argument. In the following synopses, *X* can be any character not appearing in the parameter thus bracketed.

sdefine *name X anything X*
This is like the **define** primitive, but *name* is not recognized if called with arguments.

include *file*
copy *file*
Interpolate the contents of *file*. Lines in*file* be ginning with **.EQ** or **.EN** are ignored.

ifdef *name X anything X*
If *name* has been defined by **define** (or has been automatically defined because *name* is the output driver) process *anything*; otherwise ignore *anything*.

undef *name*
Remove definition of *name*, making it undefined.

Predefined macros

GNU *eqn* supports the predefined macros offered by AT&T *eqn*: **and**, **approx**, **arc**, **cos**, **cosh**, **del**, **det**, **dot**, **dotdot**, **dyad**, **exp**, **for**, **grad**, **half**, **hat**, **if**, **inter**, **Im**, **inf**, **int**, **lim**, **ln**, **log**, **max**, **min**, **nothing**, **partial**, **prime**, **prod**, **Re**, **sin**, **sinh**, **sum**, **tan**, **tanh**, **tilde**, **times**, **union**, **vec**, ==, **!=**, +=, −>, <−, <<, >>, and "**...**". The lowercase classical Greek letters are available as **alpha**, **beta**, **chi**, **delta**, **epsilon**, **eta**, **gamma**, **iota**, **kappa**, **lambda**, **mu**, **nu**, **omega**, **omicron**, **phi**, **pi**, **psi**, **rho**, **sigma**, **tau**, **theta**, **upsilon**, **xi**, and **zeta**. Spell them with an initial capital letter (**Alpha**) or in full capitals (**ALPHA**) to obtain uppercase forms.

GNU *eqn* further defines the macros **cdot**, **cdots**, and **utilde** (all discussed above), **dollar**, which sets a dollar sign, and **ldots**, which sets three dots on the baseline.

Fonts

eqn uses up to three typefaces to set an equation: italic (oblique), roman (upright), and bold. Assign each a *groff* typeface with the primitives **gfont**, **grfont**, and **gbfont.** The defaults are the styles **I**, **R**, and **B** (applied to the current font family). The**chartype** primiti ve (see above) sets a character's type, which determines the face used to set it. The "letter" type is set in italics; others are set in roman. Use the **bold** primitive to select an (upright) bold style.

gbfont *f*
: Select *f* as the bold font. This is a GNU extension.

gfont *f*
: Select *f* as the italic font.

grfont *f*
: Select *f* as the roman font. This is a GNU extension.

Options

−−help displays a usage message, while **−v** and **−−version** show version information; all exit afterward.

−C Recognize **.EQ** and **.EN** even when followed by a character other than space or newline.

−d *xy* Specify delimiters *x* for left and *y* for right ends of equations not bracketed by **.EQ**/**.EN**. *x* and *y* need not be distinct. Any "**delim** *xy*" statements in the source file override this option.

−f *F* is equivalent to "**gfont** *F*".

−m *n* is equivalent to "**set minimum_size** *n*".

−M *dir* Search *dir* for *eqnrc* before those listed in section "Description" above.

−N Prohibit newlines within delimiters. This option allows *eqn* to recover better from missing closing delimiters.

−p *n* Set sub- and superscripts *n* points smaller than the surrounding text. This option is deprecated. *eqn* normally sets sub- and superscripts at 70% of the type size of the surrounding text.

−r Reduce the type size of subscripts at most once relative to the base type size for the equation.

−R Don't load *eqnrc*.

−s *n* is equivalent to "**gsize** *n*". This option is deprecated. *eqn* normally sets equations at the type size current when the equation is encountered.

−T *name*
: Prepare output for the device *name*. In most cases, the effect of this is to define a macro *name* with a value of **1**; *eqnrc* uses this to provide definitions appropriate for the device. However, if the specified driver is "MathML", the output is MathML markup rather than *troff* input, and *eqnrc* is not loaded at all. The default output device is **ps**.

Files

/usr/local/share/groff/1.23.0/tmac/eqnrc
: Initialization file.

MathML mode limitations

MathML is designed on the assumption that it cannot know the exact physical characteristics of the media and devices on which it will be rendered. It does not support control of motions and sizes to the same degree *troff* does.

- *eqn* customization parameters have no effect on generated MathML.
- The **special**, **up**, **down**, **fwd**, and **back** primitives cannot be implemented, and yield a MathML "<merror>" message instead.
- The **vcenter** primitive is silently ignored, as centering on the math axis is the MathML default.

- Characters that *eqn* sets extra large in *troff* mode—notably the integral sign—may appear too small and need to have their "<mstyle>" wrappers adjusted by hand.

As in its *troff* mode, *eqn* in MathML mode leaves the **.EQ** and **.EN** tokens in place, but emits nothing corresponding to **delim** delimiters. They can, however, be recognized as character sequences that begin with "$", end with "$", and do not cross line boundaries.

Caveats

Words must be quoted anywhere they occur in *eqn* input if they are not to be recognized as names of macros or primitives, or if they are to be interpreted by *troff*. These names, particularly short ones like "**pi**" and "**PI**", can collide with *troff* identifiers. For instance, the *eqn* command "**gfont PI**" does not select *groff*'s Palatino italic font for the global italic face; you must use "**gfont "PI"**" instead.

Delimited equations are set at the type size current at the beginning of the input line, not that immediately preceding the opening delimiter.

Unlike TEX, *eqn* does not inherently distinguish displayed and inline equation styles; see the **smallover** primitive above. However, macro packages frequently define **EQ** and **EN** macros such that the equation within is displayed. These macros may accept arguments permitting the equation to labeled or captioned; see the package's documentation.

Bugs

In *nroff* mode, lowercase Greek letters are rendered in roman instead of italic style.

In MathML mode, the **mark** and **lineup** features don't work. These could, in theory, be implemented with "<malignmark>" elements.

In MathML mode, each digit of a numeric literal gets a separate "<mn></mn>" pair, and decimal points are tagged with "<mo></mo>". This is allowed by the specification, but inefficient.

Examples

We first illustrate *eqn* usage with a trigonometric identity.

```
.EQ
sin ( alpha + beta ) = sin alpha cos beta + cos alpha sin beta
.EN
```

$\sin(\alpha+\beta) = \sin\alpha\cos\beta+\cos\alpha\sin\beta$

It can be convenient to set up delimiters if mathematical content will appear frequently in running text.

```
.EQ
delim $$
.EN
With a large table of logarithms in memory,
we employed the property $ln ( x y ) = ln x + ln y$ to speed the
calculation.
```

With a large table of logarithms in memory, we employed the property $\ln(xy) = \ln x + \ln y$ to speed the calculation.

See also

"Typesetting Mathematics—User's Guide" (2nd edition), by Brian W. Kernighan and Lorinda L. Cherry, 1978, AT&T Bell Laboratories Computing Science Technical Report No. 17.

The TEXbook, by Donald E. Knuth, 1984, Addison-Wesley Professional. Appendix G discusses many of the parameters from section "Customization" above in greater detail.

groff_char(7), particularly subsections "Logical symbols", "Mathematical symbols", and "Greek glyphs", documents a variety of special character escape sequences useful in mathematical typesetting.

groff(1), *troff*(1), *pic*(1), *groff_font*(5)