NAME

eqn - format equations for troff or MathML

SYNOPSIS

eqn [**-rvCNR**] [**-d** *xy*] [**-T** *name*] [**-M** *dir*] [**-f** *F*] [**-s** *n*] [**-p** *n*] [**-m** *n*] [*files . . .*]

DESCRIPTION

This manual page describes the GNU version of **eqn**, which is part of the groff document formatting system. **eqn** compiles descriptions of equations em bedded within **troff** input files into commands that are understood by **troff**. Normally, it should be invoked using the **-e** option of **groff**. The syntax is quite compatible with Unix eqn. The output of GNU **eqn** cannot be processed with Unix troff; it must be processed with GNU troff. If no files are given on the command line, the standard input is read. A filename of **-** causes the standard input to be read.

eqn searches for the file **eqnrc** in the directories given with the **-M** option first, then in **/usr/lib/groff/site-tmac**, **/usr/share/groff/site-tmac**, and finally in the standard macro directory **/usr/share/groff/1.22.3/tmac**. If it exists,**eqn** processes it before the other input files. The**-R** option prev ents this.

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).

OPTIONS

It is possible to have whitespace between a command line option and its parameter.

-d*xy* Specify delimiters *x* and *y* for the left and right end, respectively, of in-line equations. Any **delim** statements in the source file overrides this.

-C Recognize **.EQ** and **.EN** even when followed by a character other than space or newline. Also, the statement '**delim on**' is not handled specially.

-N Dont allow newlines within delimiters. This option allows **eqn** to recover better from missing closing delimiters.

-v Print the version number.

-r Only one size reduction.

-m*n* The minimum point-size is *n*. **eqn** does not reduce the size of subscripts or superscripts to a smaller size than *n*.

-T*name*
The output is for device *name*. Normally, the only effect of this is to define a macro *name* with a value of **1**; **eqnrc** uses this to provide definitions appropriate for the output device. However, if the specified device is "MathML", the output is MathML markup rather than troff commands, and **eqnrc** is not loaded at all. The default output device is **ps**.

-M*dir* Search *dir* for **eqnrc** before the default directories.

-R Dont load **eqnrc**.

-f*F* This is equivalent to a **gfont F** command.

-s*n* This is equivalent to a **gsize n** command. This option is deprecated. **eqn** normally sets equations at whatever the current point size is when the equation is encountered.

-p*n* This says that subscripts and superscripts should be *n* points smaller than the surrounding text. This option is deprecated. Normally **eqn** sets subscripts and superscripts at 70% of the size of the surrounding text.

USAGE

Only the differences between GNU **eqn** and Unix eqn are described here.

GNU **eqn** emits Presentation MathML output when invoked with the **-T MathML** option.

GNU eqn sets the input token **...** as three periods or low dots, rather than the three centered dots of classic eqn. To get three centered dots, write **cdots** or **cdot cdot cdot**.

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.

Controlling delimiters

If not in compatibility mode, **eqn** recognizes

delim on

to restore the delimiters which have been previously disabled with a call to '**delim off**'. If delimiters havent been specified, the call has no effect.

Automatic spacing

eqn gives each component of an equation a type, and adjusts the spacing between components using that type. Possible types are:

ordinary an ordinary character such as '1' or '*x*';

operator a large operator such as ' $\sum$';

binary a binary operator such as '+';

relation a relation such as '=';

opening a opening bracket such as '(';

closing a closing bracket such as ')';

punctuation a punctuation character such as ',;

inner a subformula contained within brackets;

suppress spacing that suppresses automatic spacing adjustment.

Components of an equation get a type in one of two ways.

type t e

This yields an equation component that contains *e* but that has type *t*, where *t* is one of the types mentioned above. For example, **times** is defined as

type binary (mu

The name of the type doesnt have to be quoted, but quoting protects from macro expansion.

chartype t text

Unquoted groups of characters are split up into individual characters, and the type of each character is looked up; this changes the type that is stored for each character; it says that the characters in *text* from now on have type *t*. For example,

chartype punctuation .,;:

would make the characters '.,;:' have type punctuation whenever they subsequently appeared in an equation. The type *t* can also be **letter** or **digit**; in these cases **chartype** changes the font type of the characters. See the**F onts** subsection.

New primitives

big e Enlarges the expression it modifies; intended to have semantics like CSS 'large'. In troff output, the point size is increased by 5; in MathML output, the expression uses

```
<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 display styles; **smallover** corresponds to **over** in non-display styles.

vcenter e
: This vertically centers *e* about the math axis. The math axis is the vertical position about which characters such as '+cq and '−' are centered; also it is the vertical position used for the bar of fractions. For example, **sum** is defined as

  { type operator vcenter size +5 (*S }

  (Note that 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

  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 pre-defined 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 is adjusted.

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 is not adjusted.

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 that is not the first character is treated like **opprime**.

special text e
: This constructs a new object from *e* using a troff(1) macro named *text*. When the macro is called, the string **0s** contains the output for *e*, and the number registers **0w**, **0h**, **0d**, **0skern**, and **0skew** contain the width, height, depth, subscript kern, and skew of *e*. (The *subscript kern* of an object says how much a subscript on that object should be tucked in; the *skew* of an object says how far to the right of the center of the object an accent over the object should be placed.) The macro must modify **0s** so that it outputs the desired result with its origin at the current point, and increase the current horizontal position by the width of the object. The number registers must also be modified so that they correspond to the result.

For example, suppose you wanted a construct that 'cancels' an expression by drawing a diagonal line through it.

```
.EQ
define cancel 'special Ca'
.EN
.de Ca
. ds 0s
Z'*(0s'
v'n(0du'
D'l n(0wu -n(0hu-n(0du'
v'n(0hu'
..
```

Then you could cancel an expression *e* with **cancel { e }**

Heres a more complicated construct that draws a box round an expression:

```
.EQ
define box 'special Bx'
.EN
.de Bx
. ds 0s
Z'h'1n'*(0s'
Z'
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
..
```

space n

A positive value of the integer *n* (in hundredths of an em) sets the vertical spacing before the equation, a negative value sets the spacing after the equation, replacing the default values. This primitive provides an interface to **groff**s **x** escape (but with opposite sign).

This keyword has no effect if the equation is part of a **pic** picture.

Extended primitives

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 (the value has no effect in MathML mode). Negative values are possible but have no effect. If there is more than a single value given in a matrix, the biggest one is used.

Customization

When **eqn** is generating troff markup, the appearance of equations is controlled by a large number of parameters. They have no effect when generating MathML mode, which pushes typesetting and fine motions downstream to a MathML rendering engine. These parameters can be set using the **set** command.

set p n

This sets parameter *p* to value *n*; *n* is an integer. For example,

set x_height 45

says that **eqn** should assume an x height of 0.45 ems.

Possible parameters are as follows. Values are in units of hundredths of an em unless otherwise stated. These descriptions are intended to be expository rather than definitive.

minimum_size **eqn** doesnt set anything at a smaller point-size than this. The value is in points.

fat_offset The **fat** primitive emboldens an equation by overprinting two copies of the equation horizontally offset by this amount. This parameter is not used in MathML mode; instead, fat text uses

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

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.

accent_width When **bar** or **under** is applied to a single character, the line is this long. 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 subequation that the delimiters enclose extends away from the axis.

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.

null_delimiter_space This much horizontal space is inserted on each side of a fraction.

script_space The width of subscripts and superscripts is increased by this amount.

thin_space This amount of space is automatically inserted after punctuation characters.

medium_space This amount of space is automatically inserted on either side of binary operators.

thick_space This amount of space is automatically inserted on either side of relations.

x_height The height of lowercase letters without ascenders such as 'x'.

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

default_rule_thickness This should set to the thickness of the **(ru** character, or the thickness of horizontal lines produced with the **D** escape sequence.

num1 The **over** command shifts up the numerator by at least this amount.

num2 The **smallover** command shifts up the numerator by at least this amount.

denom1 The **over** command shifts down the denominator by at least this amount.

denom2 The **smallover** command shifts down the denominator by at least this amount.

sup1 Normally superscripts are shifted up by at least this amount.

sup2 Superscripts within superscripts or upper limits or numerators of **smallover** fractions are shifted up by at least this amount. This is usually less than sup1.

sup3 Superscripts within denominators or square roots or subscripts or lower limits are shifted up by at least this amount. This is usually less than sup2.

sub1 Subscripts are normally shifted down by at least this amount.

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

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

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

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.

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.

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.

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.

big_op_spacing5 This much vertical space is added above and below limits.

baseline_sep The baselines of the rows in a pile or matrix are normally this far apart. In most cases this should be 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. In most cases this should be equal to**axis_heigh t**.

column_sep This much space is added between columns in a matrix.

matrix_side_sep This much space is added at each side of a matrix.

draw_lines If this is non-zero, lines are drawn using the **D** escape sequence, rather than with the **l** escape sequence and the **(ru** character.

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 **x**). The default value is 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 **x**). The default value is 35.

nroff If this is non-zero, then **ndefine** behaves like **define** and **tdefine** is ignored, otherwise **tdefine** behaves like **define** and **ndefine** is ignored. The default value is 0 (This is typically changed to 1 by the **eqnrc** file for the **ascii**, **latin1**, **utf8**, and **cp1047** devices.)

A more precise description of the role of many of these parameters can be found in Appendix H of *The TEXbook*.

Macros

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** command 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.

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

include *file*
copy *file*
Include the contents of *file* (**include** and **copy** are synonyms). Lines of*file* beginning 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 device) process *anything*; otherwise ignore *anything*. *X* can be any character not appearing in *anything*.

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

Besides the macros mentioned above, the following definitions are available: **Alpha**, **Beta**, ..., **Omega** (this is the same as **ALPHA**, **BETA**, ..., **OMEGA**), **ldots** (three dots on the base line), and **dollar**.

Fonts

eqn normally uses at least two fonts to set an equation: an italic font for letters, and a roman font for everything else. The existing **gfont** command changes the font that is used as the italic font. By default this is**I**. The fon t that is used as the roman font can be changed using the new **grfont** command.

grfont f
Set the roman font to *f*.

The **italic** primitive uses the current italic font set by **gfont**; the **roman** primitive uses the current roman font set by **grfont**. There is also a new**gbfon t** command, which changes the font used by the **bold** primitive. If you only use the **roman**, **italic** and **bold** primitives to changes

fonts within an equation, you can change all the fonts used by your equations just by using **gfont**, **grfont** and **gbfont** commands.

You can control which characters are treated as letters (and therefore set in italics) by using the **chartype** command described above. A type of **letter** causes a character to be set in italic type. A type of **digit** causes a character to be set in roman type.

FILES

/usr/share/groff/1.22.3/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 fine control of motions and sizes to the same degree troff does. Thus:

* **eqn** parameters have no effect on the generated MathML.
* The **special, up**, **down**, **fwd**, and **back** operations cannot be implemented, and yield a MathML '<merror>' message instead.
* The **vcenter** keyword is silently ignored, as centering on the math axis is the MathML default.
* Characters that **eqn** over troff sets extra large – 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** delimiters in place for displayed equations, but emits no explicit delimiters around inline equations. They can, however, be recognized as strings that begin with '$' and end with '$' and do not cross line boundaries.

See the **BUGS** section for translation limits specific to **eqn**.

BUGS

Inline equations are set at the point size that is current at the beginning of the input line.

In MathML mode, the **mark** and **lineup** features dont work. These could, in theory, be implemented with '<maligngroup>' 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.

SEE ALSO

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

COPYING