

# Package: mathml (via r-universe)

July 2, 2026

**Type** Package

**Title** Translate R Expressions to 'MathML' and 'LaTeX'/MathJax'

**Version** 1.9

**Date** 2026-07-01

**Maintainer** Matthias Gondan <Matthias.Gondan-Rochon@uibk.ac.at>

**Description** Translate R expressions to 'MathML' or 'MathJax'/LaTeX'  
so that they can be rendered in R markdown documents and shiny  
apps.

**License** FreeBSD

**Depends** R (>= 4.3), rolog (>= 0.9.14), knitr

**Imports** xfun (>= 0.49)

**Encoding** UTF-8

**URL** <https://github.com/mgondan/mathml>

**BugReports** <https://github.com/mgondan/mathml/issues>

**Suggests** rmarkdown, testthat

**VignetteBuilder** knitr, rmarkdown

**RoxygenNote** 7.3.3

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Matthias Gondan [aut, cre, cph] (University of Innsbruck),  
Irene Alfarone [aut] (University of Innsbruck), European  
Commission [fnd] (Erasmus+ Programme, 2019-1-EE01-KA203-051708)

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-07-02 07:00:23 UTC

**RemoteUrl** <https://github.com/cran/mathml>

**RemoteRef** HEAD

**RemoteSha** 46f721c94d3e0324d6a50a15fe887399f53b3b6d

## Contents

$\%.\%$	3
$\%dbl\downarrow\%$	3
$\%dbl\uparrow\%$	4
$\%\downarrow\%$	4
$\%==\%$	5
$\%=>\%$	5
$\%=\sim\%$	6
$\%>\%$	6
$\%<=\%$	7
$\%<=>\%$	7
$\%+\-\%$	8
$\%\prop\%$	8
$\%<\-\%$	9
$\%<\>\%$	9
$\%\sim\sim\%$	10
$\%\uparrow\%$	10
add	11
add_left	11
add_right	12
cal	12
canonical	13
decorations	13
denote	14
dfrac	15
dot	15
fname	16
fontstyles	16
frac	17
hook	17
instead	18
math	19
mathjax	19
mathml	20
mathml_preproc	21
mathout	22
name	23
omit	23
omit_left	24
omit_right	24
prod_over	25
sum_over	25

---

%.%                      *Product x \* y, shown as x dot y*

---

**Description**

Product  $x * y$ , shown as  $x \cdot y$

**Usage**

$x \cdot y$

**Arguments**

$x$	first factor
$y$	second factor

**Value**

$x * y$

---

%dbltdown%                      *Down double arrow, displayed as  $x \downarrow y$*

---

**Description**

Down double arrow, displayed as  $x \downarrow y$

**Usage**

$x \downarrow y$

**Arguments**

$x$	first element
$y$	second element

**Value**

NA, it produces a downward double arrow

---

`%dblup%`                      *Up double arrow, displayed as  $x \uparrow y$*

---

**Description**

Up double arrow, displayed as  $x \uparrow y$

**Usage**

`x %dblup% y`

**Arguments**

<code>x</code>	first element
<code>y</code>	second element

**Value**

NA, it produces a upward double arrow

---

`%down%`                      *Down arrow, presented as  $x \downarrow y$*

---

**Description**

Down arrow, presented as  $x \downarrow y$

**Usage**

`x %down% y`

**Arguments**

<code>x</code>	first element
<code>y</code>	second element

**Value**

NA, it produces a downward arrow

---

`%==%`

*Equivalence, shown as `x == y`*

---

**Description**

Equivalence, shown as `x == y`

**Usage**

`x %==% y`

**Arguments**

<code>x</code>	first argument
<code>y</code>	second argument

**Value**

`x == y`

---

`%>=%`

*Left double arrow, displayed as `x <= y`*

---

**Description**

Left double arrow, displayed as `x <= y`

**Usage**

`x %>=% y`

**Arguments**

<code>x</code>	first element
<code>y</code>	second element

**Value**

NA, it produces a left double arrow

---

`%=~%`*Congruence, shown as  $x =\sim y$* 

---

**Description**Congruence, shown as  $x =\sim y$ **Usage**`x %=~% y`**Arguments**

<code>x</code>	first argument
<code>y</code>	second argument

**Value**`x == y`, e.g., `a cong b`

---

`%->%`*Right arrow, presented as  $x \rightarrow y$* 

---

**Description**Right arrow, presented as  $x \rightarrow y$ **Usage**`x %->% y`**Arguments**

<code>x</code>	first element
<code>y</code>	second element

**Value**

NA, it produces a right arrow

---

 $\%<=%$ *Right double arrow, displayed as  $x \Rightarrow y$* 

---

**Description**

Right double arrow, displayed as  $x \Rightarrow y$

**Usage**

$x \%<=% y$

**Arguments**

$x$	first element
$y$	second element

**Value**

NA, it produces a right double arrow

---

 $\%<=>%$ *If and only if condition, displayed as  $x \Leftrightarrow y$* 

---

**Description**

If and only if condition, displayed as  $x \Leftrightarrow y$

**Usage**

$x \%<=>% y$

**Arguments**

$x$	first element
$y$	second element

**Value**

NA, it produces a double arrow double-sided

---

`%+-%`*Plus Minus, it shows x and calculates x +- y*

---

**Description**

Plus Minus, it shows x and calculates x +- y

**Usage**`x %+-% y`**Arguments**

<code>x</code>	first term
<code>y</code>	second term

**Value**`c(x - y, x + y) x plus min y`

---

`%prop%`*Proportional, shown as x o< y*

---

**Description**

Proportional, shown as x o&lt; y

**Usage**`x %prop% y`**Arguments**

<code>x</code>	first argument
<code>y</code>	second argument

**Value**

NA

---

%<-% *Left arrow, presented as x <- y*

---

**Description**

Left arrow, presented as x <- y

**Usage**

x %<-% y

**Arguments**

x	first element
y	second element

**Value**

NA, it produces a left arrow

---

%<->% *Double sided arrow, presented as x <-> y*

---

**Description**

Double sided arrow, presented as x <-> y

**Usage**

x %<->% y

**Arguments**

x	first element
y	second element

**Value**

NA, it produces a double sided arrow

---

`%~~%`*Approximate equality, shown as  $x \sim y$* 

---

**Description**

Approximate equality, shown as  $x \sim y$

**Usage**

`x %~~% y`

**Arguments**

<code>x</code>	first argument
<code>y</code>	second argument

**Value**

The result of `isTRUE(all.equal(x, y))`

---

`%up%`*Up arrow, presented as  $x \uparrow y$* 

---

**Description**

Up arrow, presented as  $x \uparrow y$

**Usage**

`x %up% y`

**Arguments**

<code>x</code>	first element
<code>y</code>	second element

**Value**

NA, it produces an upward arrow

---

add	<i>add</i>
-----	------------

---

**Description**

This is a function that allows the user to highlight the mistakes, in particular an extra element in a list

**Usage**

add(expr)

**Arguments**

expr            expression

**Value**

expr , e.g., highlights a + b from a + b

---

add_left	<i>add_left</i>
----------	-----------------

---

**Description**

This is a function that allows the user to highlight the mistakes, in particular the redundancies in the left-hand side of the expression.

**Usage**

add\_left(expr)

**Arguments**

expr            expression

**Value**

expr e.g., highlights a + from a + b

---

add_right	<i>add_right</i>
-----------	------------------

---

**Description**

This is a function that allows the user to highlight the mistakes, in particular the redundancies in the right-hand side of the expression.

**Usage**

```
add_right(expr)
```

**Arguments**

expr	expression
------	------------

**Value**

expr , e.g., highlights + b from a + b

---

cal	<i>Calligraphic font</i>
-----	--------------------------

---

**Description**

Calligraphic font

**Usage**

```
cal(x)
```

**Arguments**

x	an R symbol. This function is used to render the content in calligraphic font in MathJax. In MathML, script font is used.
---	---

**Value**

The function cal is a wrapper for the identity function.

**See Also**

[identity\(\)](#)

**Examples**

```
mathjax(quote(K %in% cal(K)))
```

---

canonical	<i>Canonicalize an R call: Reorder the function arguments</i>
-----------	---

---

**Description**

Canonicalize an R call: Reorder the function arguments

**Usage**

```
canonical(term = quote(`%in%`(table = Table, x = X)), drop = TRUE)
```

**Arguments**

term	an R call.
drop	whether to drop the argument names or not

**Value**

The R function, with arguments rearranged

**Examples**

```
canonical(term=quote(`%in%`(table=Table, x=X)))
```

---

decorations	<i>Identity functions for different decorations</i>
-------------	---

---

**Description**

Identity functions for different decorations

**Usage**

```
roof(x)
```

```
boxed(x)
```

```
cancel(x)
```

```
phantom(x)
```

```
prime(x)
```

```
tilde(x)
```

over(x)

under(x)

underover(x)

hyph(x)

color(x)

### Arguments

x                    the expression to render

### Value

x

### Examples

```
roof(1) + mean(2) + boxed(3) + cancel(4) + phantom(5) + prime(6) + tilde(7)
```

```
mathml(quote(roof(b) + mean(X) + boxed(3) + cancel(4) + phantom(5)))
```

---

denote	<i>denote This is a function that allows the user to insert abbreviations in the formula, explain them and make the needed computations</i>
--------	---

---

### Description

denote This is a function that allows the user to insert abbreviations in the formula, explain them and make the needed computations

### Usage

```
denote(abbr, expr, info)
```

### Arguments

abbr	Abbreviation used in the text to refer to the calculation, for example 's_p' for the pooled variance.
expr	Expression: calculations to be made in order to obtain the value to which the abbreviation refers to.
info	Information: Explanation of the formula used to provide the value of the abbreviation. e.g. 'the pooled variance'

### Value

expr e.g., x denotes  $a^2 + b$

---

dfrac	<i>Division displayed as large fraction</i>
-------	---

---

**Description**

Division displayed as large fraction

**Usage**

```
dfrac(e1, e2)
```

**Arguments**

e1	numerator
e2	denominator

**Value**

$e1 / e2$

**See Also**

[frac\(\)](#), [over\(\)](#)

---

dot	<i>Multiplication</i>
-----	-----------------------

---

**Description**

Multiplication

**Usage**

```
dot(e1, e2)  
nodot(e1, e2)  
times(e1, e2)
```

**Arguments**

e1	numerator
e2	denominator

**Value**

$e1 * e2$

fname                      *Return function body*

---

**Description**

Return function body

**Usage**

fname(fname, body)

**Arguments**

fname	not clear
body	not clear

**Value**

body

---

fontstyles                      *Identity functions for different font styles*

---

**Description**

Identity functions for different font styles

**Usage**

plain(x)

italic(x)

bold(x)

**Arguments**

x	the expression to render
---	--------------------------

**Value**

x

**Examples**

```
plain(1) + bold(2) + italic(3)
```

```
mathml(term=quote(plain(abc) + bold(def) + italic(ghi)))
```

---

frac	<i>Division displayed as fraction</i>
------	---------------------------------------

---

**Description**

Division displayed as fraction

**Usage**

```
frac(e1, e2)
```

**Arguments**

e1	numerator
e2	denominator

**Value**

e1 / e2

---

hook	<i>Hook for custom symbols</i>
------	--------------------------------

---

**Description**

hook(term, display) hook\_fn(fn) unhook(term) hooked(term)

**Usage**

```
hook(term, display = NULL, quote = TRUE, as.olog = TRUE)
```

```
unhook(term, quote = TRUE, as.olog = TRUE)
```

```
hooked(term)
```

```
hook_fn(fn)
```

**Arguments**

term	an R call or symbol/number. This is the expression to replace.
display	an R call or symbol/number. This is shown instead of <i>term</i> .
quote	(default is TRUE) indicates that <i>term</i> and <i>display</i> should be quoted.
as.olog	(default is TRUE) indicates that simplified quasi-quotation is to be used.
fn	a custom function. The name of <i>fn</i> is replaced by its function body.

**Value**

hook and unhook return TRUE on success. hooked returns the hooked expression or FALSE on failure.

**Examples**

```
hook(t0, subscript(t, 0))
hooked(quote(t0))
mathml(quote(t0))
hook(term=quote(t0), display=quote(superscript(t, 0)), quote=FALSE)
mathml(quote(t0))
unhook(t0)
mathml(quote(t0))
square <- function(x) {x^2} ; hook_fn(square)
```

---

instead

*instead*

---

**Description**

This is a function that allows the user to highlight the mistakes, in particular adds a curly bracket under the wrong term and it provides the correct solutions.

**Usage**

```
instead(inst, of)
```

**Arguments**

inst	the wrong term
of	the correct term

**Value**

inst

**Examples**

```
1 + instead(2, 3)
mathml(term=quote(1 + instead(2, 3)))
```

---

math	<i>Adds the class "math" to the object for knitr output via mathout()</i>
------	---

---

**Description**

Adds the class "math" to the object for knitr output via `mathout()`

**Usage**

```
math(term, flags = NULL)
```

**Arguments**

term	an R call or symbol/number. This function translates <i>term</i> into a LaTeX/MathJax string.
flags	(default NULL) list of flags that control the translation

**Value**

*term* with additional class "math" and *flags* as attributes.

**See Also**

[mathml\(\)](#), [mathjax\(\)](#), [mathout\(\)](#)

**Examples**

```
math(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

---

mathjax	<i>Mathjax output</i>
---------	-----------------------

---

**Description**

Mathjax output

**Usage**

```
mathjax(
  term = quote((a + b)^2L == a^2L + 2L * a * b + b^2L),
  flags = NULL,
  env = globalenv()
)
```

**Arguments**

term	an R call or symbol/number. This function translates <i>term</i> into a LaTeX/MathJax string.
flags	(default NULL) list of flags that control the translation
env	(default globalenv()) The R environment in which <code>r_eval</code> is being executed (see vignette for details, "Ringing back to R").

**Details**

In some functions, the Prolog code may ring back R, for example, to find the names of function arguments. For example (see vignette), when rendering the call `integrate(g, lower=0L, upper=Inf)` as  $\int_0^{\infty} g(x) dx$ , Prolog needs to know that the function `g` is a function of `x`. The Prolog rule then searches for the `formalArgs` of `g` in the environment `env`.

**Value**

A string with the MathJax representation of *term*.

**See Also**

[mathml\(\)](#)

**Examples**

```
mathjax(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

---

mathml

*MathML output*

---

**Description**

MathML output

**Usage**

```
mathml(
  term = quote((a + b)^2L == a^2L + 2L * a * b + b^2L),
  flags = NULL,
  env = globalenv()
)
```

**Arguments**

term	an R call or symbol/number. This function translates <i>term</i> into a MathML string.
flags	(default NULL) list of flags that control the translation. This includes "context" settings such as <code>error("ignore")</code> , or a default number of decimal places for numeric output.
env	(default globalenv()) The R environment in which <code>r_eval</code> is being executed.

**Details**

In some functions, the Prolog code may ring back R, for example, to find the names of function arguments. For example (see vignette), when rendering the call `integrate(g, lower=0L, upper=Inf)` as  $\int_0^\infty g(x) dx$ , Prolog needs to know that the function `g` is a function of `x`. The Prolog rule then searches for the `formalArgs` of `g` in the environment `env`.

**Value**

A string with the MathML representation of `term`.

**See Also**

[mathjax\(\)](#)

**Examples**

```
mathml(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
mathml(term=3.14159265, flags=list(round=3L))
mathml(term=3.14159265, flags=list(quote(round(3L))))
```

---

mathml\_preproc

*Map R operators to their respective Prolog counterparts*

---

**Description**

Map R operators to their respective Prolog counterparts

**Usage**

```
mathml_preproc(query = quote(5%%2))
```

**Arguments**

`query` an R call or symbol/number. This function translates components of `query` into their respective counterparts from Prolog

**Value**

The translated query

**See Also**

[mathjax\(\)](#), [mathml\(\)](#)

**Examples**

```
mathml_preproc(quote(5 %% 2))
```

---

`mathout`*MathML or MathJax output, depending on the knitr context*

---

**Description**

MathML or MathJax output, depending on the knitr context

**Usage**

```
mathout(term, flags = NULL, env = parent.frame())
```

```
inline(term, flags = NULL, env = parent.frame())
```

**Arguments**

<code>term</code>	an R call or symbol/number. This function translates <i>term</i> into a LaTeX/MathJax string.
<code>flags</code>	(default NULL) list of flags that control the translation
<code>env</code>	(default <code>parent.frame()</code> ) The R environment in which <code>r_eval</code> is being executed (see vignette for details, "Ringing back to R").

**Details**

This function checks `knitr::is_html_output()` and `knitr::is_html_output()` and invokes the respective function `mathml()` or `mathjax()`. Outside of knitr context, MathML is returned, and a warning is given.

**Value**

A string with the MathML or MathJax representation of *term*.

**See Also**

[mathml\(\)](#), [mathjax\(\)](#), [inline\(\)](#)

**Examples**

```
mathout(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

```
inline(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

---

name	<i>Add a name attribute to an element (most often, an R function)</i>
------	---

---

**Description**

Add a name attribute to an element (most often, an R function)

**Usage**

```
name(x, name)
```

**Arguments**

x	an R object, e.g., an R function
name	the name of the object/function

**Value**

The object with the name attribute

**Examples**

```
f <- function(x) {sin(x)}  
mathjax(call("integrate", name(f, "sin"), 0L, 2L*pi))
```

---

omit	<i>omit</i>
------	-------------

---

**Description**

This is a function that allows the user to highlight the mistakes, in particular the omission of an element from a list.

**Usage**

```
omit(expr)
```

**Arguments**

expr	expression
------	------------

**Value**

NULL e.g., remove a + b from a + b

---

omit_left	<i>omit_left</i> This is a function that allows the user to highlight the mistakes, in particular the omissions in the left-hand side of the expression
-----------	---

---

**Description**

omit\_left This is a function that allows the user to highlight the mistakes, in particular the omissions in the left-hand side of the expression

**Usage**

omit\_left(expr)

**Arguments**

expr	The expression, e.g. a + b
------	----------------------------

**Value**

substitute(expr)[[3]], e.g., b from a + b

---

omit_right	<i>omit_right</i> This is a function that allows the user to highlight the mistakes, in particular the omissions in the right-hand side of the expression
------------	---

---

**Description**

omit\_right This is a function that allows the user to highlight the mistakes, in particular the omissions in the right-hand side of the expression

**Usage**

omit\_right(expr)

**Arguments**

expr	expression
------	------------

**Value**

substitute(expr)[[2]], e.g., a from a + b

---

prod_over	<i>product over a range. On the R side, this function just returns the product of the first argument, but allows for decorations.</i>
-----------	---

---

**Description**

product over a range. On the R side, this function just returns the product of the first argument, but allows for decorations.

**Usage**

```
prod_over(x, from, to)
```

**Arguments**

x	the object to be multiplied
from	decoration for $\prod_{i=from}^{to} x_i$
to	decoration for $\prod_{i=from}^{to} x_i$

**Value**

The function returns  $\prod(x)$

**See Also**

[prod\(\)](#), [sum\\_over\(\)](#)

**Examples**

```
mathjax(quote(prod_over(x[i], i=1L, N)))
```

---

sum_over	<i>sum over a range. On the R side, this function just returns the first argument, but allows for decorations.</i>
----------	--

---

**Description**

sum over a range. On the R side, this function just returns the first argument, but allows for decorations.

**Usage**

```
sum_over(x, from, to)
```

**Arguments**

x	the object to be summed
from	decoration for $\sum_{\text{from}}^{\text{to}} x_i$
to	decoration for $\sum_{\text{from}}^{\text{to}} x_i$

**Value**

The function returns  $\text{sum}(x)$

**See Also**

[sum\(\)](#), [prod\\_over\(\)](#)

**Examples**

```
mathjax(quote(sum_over(x[i], i=1L, N)))
```

# Index

$\%+-\%$ , 8  
 $\%->\%$ , 6  
 $\%.\%$ , 3  
 $\%<->\%$ , 9  
 $\%<-\%$ , 9  
 $\%<=>\%$ , 7  
 $\%<=\%$ , 7  
 $\%==\%$ , 5  
 $\%=>\%$ , 5  
 $\%-\sim\%$ , 6  
 $\%\sim\sim\%$ , 10  
 $\%dbl\down\%$ , 3  
 $\%dbl\up\%$ , 4  
 $\%down\%$ , 4  
 $\%prop\%$ , 8  
 $\%up\%$ , 10

add, 11  
add\_left, 11  
add\_right, 12

bold (fontstyles), 16  
boxed (decorations), 13

cal, 12  
cancel (decorations), 13  
canonical, 13  
color (decorations), 13

decorations, 13  
denote, 14  
dfrac, 15  
dot, 15

fname, 16  
fontstyles, 16  
frac, 17  
frac(), 15

hook, 17  
hook\_fn (hook), 17

hooked (hook), 17  
hyph (decorations), 13

identity(), 12  
inline (mathout), 22  
inline(), 22  
instead, 18  
italic (fontstyles), 16

math, 19  
mathjax, 19  
mathjax(), 19, 21, 22  
mathml, 20  
mathml(), 19–22  
mathml\_preproc, 21  
mathout, 22  
mathout(), 19

name, 23  
nodot (dot), 15

omit, 23  
omit\_left, 24  
omit\_right, 24  
over (decorations), 13  
over(), 15

phantom (decorations), 13  
plain (fontstyles), 16  
prime (decorations), 13  
prod(), 25  
prod\_over, 25  
prod\_over(), 26

roof (decorations), 13

sum(), 26  
sum\_over, 25  
sum\_over(), 25

tilde (decorations), 13

times (dot), [15](#)

under (decorations), [13](#)

underover (decorations), [13](#)

unhook (hook), [17](#)