The math-shorthandz Package

Smart Shorthands for Math Typesetting

Tiago Pomella Lobo

November 24, 2026 v1.1.0

Abstract

The math-shorthandz package has shorthand macros for typesetting the most common mathematical expressions in LaTeX. It provides compact commands to construct symbols with features such as multi-underlines, pre- and succeeding scripts, fractions, differentials and derivatives, and integrals, with highly customizable positioning and (auto-)scaling. All commands are safe in text, and distinguish their customization between display, inline, script and scriptscript contexts.

Contents

1 Introduction **2**
- 1.1 About this documentation . . . 2
- 1.2 License and support . . . 2

2 User Interface **2**
- 2.1 `\zpunct` — punctuation at the end of a display line . . . 3
- 2.2 `\zline` — arbitrary underline stacking . . . 4
- 2.3 `\zfrac` — flexible fraction . . . 5
- 2.4 `\zscale` — uniform content scaling . . . 6
- 2.5 `\zdot` — bold dot-product operator . . . 8
- 2.6 `\zarg` — scaled function argument . . . 9
- 2.7 `\zdiff` — differential or derivative . . . 10
- 2.8 `\zcond` — bar to represent evaluation conditions . . . 12
- 2.9 `\zscript` — four-way sub/superscripts . . . 13
- 2.10 `\zprime` — prime accents . . . 17
- 2.11 `\zlim` — limit with variable and value . . . 20
- 2.12 `\zint` — configurable integral . . . 22

3 Package options **27**
- 3.1 Global options . . . 27
- 3.2 Local options . . . 27
- 3.3 Aliases . . . 28

4 Combined Examples **31**
- 4.1 Fundamental theorem of calculus . . . 31
- 4.2 Tensor with four-way scripts and evaluation bar . . . 31
- 4.3 Divergence theorem . . . 31
- 4.4 Multi-dimensional integration . . . 31
- 4.5 Gaussian integral . . . 31
- 4.6 Circulation of a vector field . . . 32
- 4.7 Arc length of a parametrised curve . . . 32
- 4.8 Moment of inertia of spherical shell . . . 32
- 4.9 Heat equation solution . . . 32
- 4.10 Beer–Lambert law . . . 33

5 Change History **34**

1 Introduction

Classical LaTeX math commands such as `\int`, `\frac`, and `\underbrace` are powerful but low-level. Writing readable source for an equation requires manual spacing kerns, sizing primitives, and mode switches that clutter the source and make it hard to refactor. As an alternative, I have been using several macros intermittently since 2020 to hide much of this complexity. Now, I have merged and expanded these macros into the math-shorthandz package that introduces many semantic shorthands. In this new version, the consolidated commands handle both display and inline modes, respect the ambient math style, and expose their tunable parameters as package options so the same source adapts to different fonts and house styles without changes to individual call sites. You can see some examples below:

```
Show case with Equation~\ref{eq:example}:
\begin{equation}
  \label{eq:example}
  \zint[0][T] f\zarg{t}\zdiff+{t} = \zfrac[\beta+1]
  \zpunct{.}
\end{equation}
```

Show case with Equation 1:

$$\int_0^T f(t)\ \mathrm{d}t = \frac{1}{(\beta+1)}\,. \tag{1}$$

The package is built on expl3 and xparse, which are part of the LaTeX kernel since 2020 and require no extra `\usepackage` declarations. The integral commands use XITS Math glyphs for the integral symbols (loaded automatically at `\begin{document}`); the remainder of the math font is left to the user. I personally like:

```
\setmathfont{TeX Gyre DejaVu Math}[Scale=0.85]
```

The package automatically loads amsmath, unicode-math, graphicx, nicefrac, xkeyval. Because unicode-math is loaded, **the document engine must be** LuaLaTeX or XeLaTeX. Import it with:

```
\usepackage[⟨options⟩]{math-shorthandz}
```

The package name has a playful ”z” ending to emphasize the unified naming style for all commands, that also distinguishes them (*e.g.* `\zdot` instead of `\cdot`). The package was originally named zmath, but was renamed upon its first public release to address requests from CTAN and TeX Live.

1.1 About this documentation

The local macro `\nl` is used to add line breaks in the example boxes, so it has been colored grey to reduce visual impact. Every option’s key name is prefixed with the full public command name it belongs to (*e.g.* `zint-`, `zdiff-`), so key names are unambiguous even when taken out of context. The environment `macroTable` typesets the captioned option table that follows the syntax of each command, with standard columns ”Option”, ”Default” and ”Description”. Other headers can be optionally passed, and the starred form `macroTable*` prduces example tables with the columns ”Source”, ”Output” and ”Meaning”.

1.2 License and support

Copyright © 2020–2026 Tiago Pomella Lobo. This work may be distributed and/or modified under the conditions of the LaTeX Project Public License, version 1.3c or later (`https://www.latex-project.org/lppl.txt`), and has the maintenance status ”author-maintained”. Source code, issues and releases: `https://github.com/tplobo/math-shorthandz`.

2 User Interface

The options for each command are listed in the tables of their own subsection. Some brief examples are shown using `\setkeys` to change them, but more details on the different ways to set them can be found in Section 3.

2.1 \zpunct — punctuation at the end of a display line

\zpunct The \zpunct command provides a semantic hook to end a line of a mathematical display with a punctuation mark *⟨punct⟩*. Rather than dropping the token right at the end of the math expression—which can cause awkward line breaks or misaligned punctuation tracking—this utility moves it to the right end of the line, where the equation number goes. It works with and without equation numbers. It has no package options.

```
\zpunct{⟨punct⟩}
```

In a numbered equation, the punctuation rests immediately before the number, separated from it by a space. In an unnumbered equation (*e.g.* `equation*`, `gather*`, `flalign*`, …), the punctuation is flushed right to where the numbering would stand. This is done with the `\tag*` command of `amsmath`, so `\zpunct` cannot be combined with `\tag` in the same unnumbered equation. A row suppressed with `\notag` or `\nonumber` inside a numbered environment discards the punctuation. Place the command in the line it ends, before a newline command (*e.g.* \\).

The following example punctuates the equations in a numbered block:

```
Equations \ref{eq:energy} and \ref{eq:force}
show important results:
\begin{gather}
  \label{eq:energy}
  E = mc^2                          \zpunct{,}
  \\
  \label{eq:force}
  \vec{F} = m \cdot \vec{a}         \zpunct{.}
\end{gather}
```

Equations 2 and 3 show important results:

$$E = mc^2 \quad , (2)$$
$$\vec{F} = m \cdot \vec{a} \quad . (3)$$

while this example shows how to complement text within an unnumbered block, where punctuation is flushed right without requiring the numbering:

```
The following equations show important results:
\begin{flalign*}
  & E = mc^2                         \zpunct{, and}
  \\
  & \vec{F} = m \cdot \vec{a}        \zpunct{.}
&\end{flalign*}
```

The following equations show important results:

$$E = mc^2 \quad \text{, and}$$
$$\vec{F} = m \cdot \vec{a} \quad .$$

2.2 \zline — arbitrary underline stacking

\zline The \zline utility structurally attaches an arbitrary ⟨count⟩ of horizontal underline rules beneath a target ⟨symbol⟩. By default, omitting the optional parameter draws a single line. The execution engine dynamically computes the natural horizontal bounding box of the input ⟨symbol⟩, ensuring that the rule widths match the content exactly without leaking into adjacent characters. It is safe for direct deployment in both text and math modes. Its options are listed in Table 1.

```
\zline[⟨count⟩]{⟨symbol⟩}
```

Table 1: Options for \zline, \zvec and \zmat.

Option	Default	Description
zline-depth	1pt	Vertical gap from the symbol's bottom edge to the first rule (Figure 1).
zline-thickness	0.4pt	Absolute typographic thickness of each individual underline rule.
zline-separation	0.9pt	Vertical gap between consecutive stacked rules.

The macro was inspired by the popular ulem package, but alters some of its functionalities: instead of line weights that depend on ambient font sizes (which may cause layout collisions), \zline enforces a fixed baseline and thickness alignment. The absolute vertical layout dimensions of rule thickness, clearances from the symbol's bottom edge, and relative gaps between stacked lines are governed by the zline-thickness, zline-depth, and zline-separation package options (Figure 1). The symbol box is typeset once with \mathsurround suppressed.

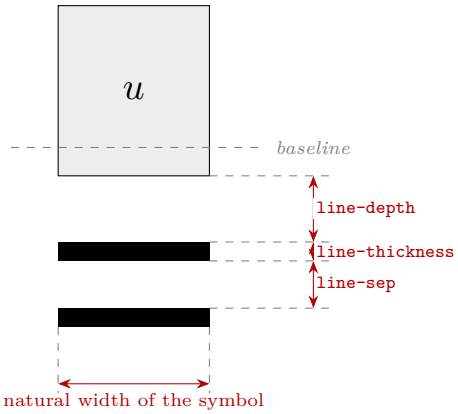


Figure 1: Schematic of the \zline options (here two rules, as in \zmat). The shaded box is the symbol's box; the rules are drawn filled, with the same width. Distances are exaggerated for legibility.

\zvec For convenience in physics and engineering notation, the package provides the secondary shorthand macros
\zmat \zvec and \zmat to standardize vector and matrix underline conventions, as an alternative to bold fonts, \vec, \overleftrightarrow and others, that can lead to dense notation. Both commands call \zline, generating one and two underlines respectively.

```
\zvec{⟨symbol⟩}      \zmat{⟨symbol⟩}
```

The following example compares \zvec and \zmat with the ulem and accent alternatives, and shows a triple underline and the difference between underlining an expression and a symbol:

```
$\zvec{u} = \vec{u} \ne \uline{u}$                % zvec vs ulem command
\nl
$\zmat{A} = \overleftrightarrow{A} \ne \uuline{A}$  % zmat vs ulem command
\nl
\zline[3]{W}                                      % triple underline
\nl
$\zline{x_b^2} \ne \zline{x}_b^2$                 % composition
```

$\underline{u} = \vec{u} \neq \underline{u}$

$\underline{\underline{A}} = \overleftrightarrow{A} \neq \underline{\underline{A}}$

$\underline{\underline{\underline{W}}}$

$\underline{x_b^2} \neq \underline{x}_b^2$

2.3 \zfrac — flexible fraction

\zfrac The \zfrac utility typesets a fraction whose visual style and argument parenthesisation are controlled independently, by stars and by the choice of delimiter around each argument. If only one argument is supplied, it is treated as the denominator and the numerator defaults to 1. The entire output is always wrapped in \ensuremath, so \zfrac is safe in text, inline and display mode to contain math symbols (*e.g.* Greek letters, radicals). It has no package options.

```
\zfrac**[⟨num⟩]{⟨num⟩}[⟨den⟩]{⟨den⟩}
```

Starring:

- **none (-)** — standard \frac layout.
- **one (*)** — oblique \nicefrac layout, suited to inline use.
- **two (**)** — slash notation, ⟨num⟩/⟨den⟩.

Each argument may be given in braces ({⟨x⟩}, plain) or in brackets ([⟨x⟩], parenthesised, *i.e.* wrapped in \left(… \right)). The two delimiter choices are independent, so *e.g.* only the denominator may be parenthesised while the numerator is left plain. The following table lists every delimiter combination:

Source	Inline	Display	Meaning
\zfrac{n}{d}	$\frac{n}{d}$	$\frac{n}{d}$	plain numerator, plain denominator
\zfrac{n}[d]	$\frac{n}{(d)}$	$\frac{n}{(d)}$	parenthesised denominator
\zfrac[n]{d}	$\frac{(n)}{d}$	$\frac{(n)}{d}$	parenthesised numerator
\zfrac[n][d]	$\frac{(n)}{(d)}$	$\frac{(n)}{(d)}$	both parenthesised
\zfrac{d}	$\frac{1}{d}$	$\frac{1}{d}$	single arg $\implies$ numerator = 1
\zfrac[d]	$\frac{1}{(d)}$	$\frac{1}{(d)}$	parenthesised, with numerator = 1

and here are some accompanying examples:

```
\zfrac{\alpha}{\beta}                  % standard frac
\nl
\zfrac*{x}{y}                          % nicefrac
\nl
\zfrac**{x}{y}                         % slash
\nl
$\zfrac{\alpha}[\beta+1]$              % parenthesise denominator
\nl
$\zfrac[\alpha+1]{\beta}$              % parenthesise numerator
\nl
$\zfrac{\lambda}$                      % inverse
\nl
$e^{-\zfrac{x}} \ne e^{-\zfrac*{x}}$   % context aware
```

$\frac{\alpha}{\beta}$

x/y

x/y

$\frac{\alpha}{(\beta+1)}$

$\frac{(\alpha+1)}{\beta}$

$\frac{1}{\lambda}$

$e^{-\frac{1}{x}} \neq e^{-1/x}$

2.4 `\zscale` — uniform content scaling

`\zscale` The `\zscale` utility scales an arbitrary ⟨*math*⟩ token vector uniformly by a given ⟨*factor*⟩. If the optional modifier is omitted, the engine defaults to the package option of the math style in which `\zscale` is called (`zscale-display`, `zscale-text`, `zscale-script` or `zscale-scriptscript`); an explicit ⟨*factor*⟩ overrides the factor in whichever style the call sits. With `zscale-absolute` set to `true` (the default), the content is set to text size and the factor is subsequently applied. With `false`, the content is set in the math style of the surrounding context *before* being scaled, so scaling compounds with TEX's own shrinking factor. Its options are listed in Table 2.

```
\zscale[⟨factor⟩]{⟨math⟩}
```

Table 2: Options for `\zscale`.

Option	Default	Description
`zscale-absolute`	true	`true`: the factors multiply the content set in the *normal* size (display style in display context, text style otherwise). `false`: the factors multiply the content set in the ambient style, as TEX would, so both factors stack (all factors 1 gives plain LATEXscaling). In both cases, `zscale-scriptscript` is set as the lower bound and nested calls do not scale below that (unless set explicitly).
`zscale-display`	1	Scale factor applied by `\zscale` in display style.
`zscale-text`	1	As `zscale-display`, in text style.
`zscale-script`	0.7	As `zscale-display`, in script style.
`zscale-scriptscript`	0.6	As `zscale-display`, in scriptscript style.

The macro is inherently safe for direct deployment across both math configurations and ambient text strings via an internal `\ensuremath` boundary context. The following example compares factors, call contexts and a plain subscript:

```
$A_{XYZ} \ne A_{\zscale{XYZ}}$          % default scaling
\nl
$A_{XYZ} \ne A_{\zscale[0.6]{XYZ}}$     % 60% scaling
\nl
$e^{A_{XYZ}} \ne e^{A_{\zscale{XYZ}}}$  % context-aware scaling
```

$$A_{XYZ} \neq A_{XYZ}$$
$$A_{XYZ} \neq A_{XYZ}$$
$$e^{A_{XYZ}} \neq e^{A_{XYZ}}$$

Since a scaled box is invisible to TEX's math styles, an explicit guard is set for nested calls (*e.g.* `\zscale{\zscale{{⟨math⟩}}}`) to avoid illegible symbols. The size of scriptscript style, *i.e.* `zscale-scriptscript`, is a lower bound for scaling. However, a factor that is already smaller than that (passed as an argument or set as an option) is still accepted:

```
\[
  X \to \zscale{X} \ne \zscale[0.4]{X}          % display
\]
$ X \to \zscale{X} \ne \zscale[0.4]{X} $        % text
\nl
$ X^X \to X^{\zscale{X}} \ne X^{\zscale[0.4]{X}} $   % script
\nl
$ X^{X^X} \to X^{X^{\zscale{X}}} $              % scriptscript
```

$$X \to X \neq {}_{x}$$
$$X \to X \neq {}_{x}$$
$$X^X \to X^X \neq X^{x}$$
$$X^{X^X} \to X^{X^X}$$

Other commands that scale content using `\zscale` have four scale options, one per math style, and a switch analogous to `zscale-absolute`, so scaling choices remain independent; their keys are written `<command>-scale-<style>` and `<command>-scale-absolute`. The reference is the style of the call for `\zscale`, as well as `\zdot`, `\zarg` and `\zprime`. But the reference is the script style of the nucleus, one level below it and never below scriptscript, for `\zcond`, `\zscript`, `\zlim` and `\zint`.

Choosing scaling factors. Factors can be chosen by deciding how large the output should be compared to standard TEX behavior. If that reference size relative to the normal size is r, and the wanted fraction is k, then:

- when `zscale-absolute` is set to `true`, the factor multiplies the normal size, so the output is scaled to $k \times r$;
- when `zscale-absolute` is set to `false`, the content is already set at the reference size, so the output is scaled to k.

Most math fonts shrink script and scriptscript style to about 70% and 50% of the normal size, *i.e.* $r \approx 0.7$ and $r \approx 0.5$. So, the factor for a script-size symbol must be $k \times r$ in absolute mode, or k in relative mode; *e.g.* $0.9 \times 0.7 = 0.63$ in absolute mode (or 0.9 in relative mode) makes it 90% of the standard script-size symbol. The following examples show relative scaling set with `\setkeys`. They compare unscaled output against the default scaling (absolute), against a relative scaling with unchanged factors, and against a relative scaling with all factors changed to 0.8:

```
$X \; X^X \; X^{X^X}$                                    % TeX
\nl
$\zscale{X} \; X^{\zscale{X}} \; X^{X^{\zscale{X}}}$    % zscale
\nl[3]
{
  \setkeys{shorthand}{zscale-absolute=false}
  $\zscale{X} \; X^{\zscale{X}} \; X^{X^{\zscale{X}}}$  % relative
}
\nl[10]
{
  \setkeys{shorthand}{
    zscale-absolute=false,
    zscale-display=0.8,
    zscale-text=0.8,
    zscale-script=0.8,
    zscale-scriptscript=0.8
  }
  $\zscale{X} \; X^{\zscale{X}} \; X^{X^{\zscale{X}}}$  % relative, k = 0.8
}
```

$X \; X^X \; X^{X^X}$

$X \; X^X \; X^{X^X}$

$X \; X^x \; X^{X^x}$

$X \; X^X \; X^{X^X}$

2.5 \zdot — bold dot-product operator

\zdot The \zdot command produces a visually heavier centred dot for use as a vector or tensor dot-product operator, offering a clear typographic distinction from the standard \cdot. The \zdot command takes no arguments. Place it between operands exactly as you would \cdot or \times. Its options are listed in Table 3.

```
\zdot
```

Table 3: Options for \zdot.

Option	Default	Description
zdot-scale-absolute	true	As zscale-absolute, for \zdot.
zdot-scale-display	1.8	Scale factor applied to \cdot when \zdot is typeset in display style. Increase for a heavier dot, decrease to bring it closer to \cdot.
zdot-scale-text	1.8	As zdot-scale-display, in text style.
zdot-scale-script	1.4	As zdot-scale-display, in script style (*e.g.* inside an exponent).
zdot-scale-scriptscript	1	As zdot-scale-display, in scriptscript style.

The glyph is constructed by scaling \cdot with the zdot-scale-⟨*style*⟩ option of the ambient math style (see zscale) and moving it vertically so that it sits exactly on the axis of the ambient style, which compensates for the baseline drift that \scalebox introduces in math mode, in any scale. The command is declared as \mathbin so that surrounding spacing follows binary-operator conventions. If the dot looks too close in weight to \cdot, or too heavy inside exponents, adjust the scale of the corresponding style.

The following example contrasts \zdot with the regular \cdot between vectors and shows the output in different font sizes:

```
\[
  W = \vec{F} \zdot \vec{d}                                  % display
\]
\nl
$u \cdot v \ne \vec{u} \zdot \vec{v}$                        % text
\nl
$e^{u \cdot v} \ne e^{\vec{u} \zdot \vec{v}}$                % script
\nl
$e^{A_{u \cdot v}} \ne e^{A_{\vec{u} \zdot \vec{v}}}$ % scriptscript
```

$$W = \vec{F} \mathbin{\cdot} \vec{d}$$

$u \cdot v \ne \vec{u} \mathbin{\cdot} \vec{v}$

$e^{u \cdot v} \ne e^{\vec{u} \mathbin{\cdot} \vec{v}}$

$e^{A_{u \cdot v}} \ne e^{A_{\vec{u} \mathbin{\cdot} \vec{v}}}$

2.6 \zarg — scaled function argument

\zarg The \zarg utility typesets an expression ⟨*expr*⟩ inside parentheses, scaled, and inserting a small kern immediately before the opening delimiter. This is useful to distinguish function arguments from other uses of parentheses. Its options are listed in Table 4.

```
\zarg{⟨expr⟩}
```

Table 4: Options for \zarg.

Option	Default	Description
zarg-scale-absolute	true	As zscale-absolute, for \zarg.
zarg-scale-display	0.9	Scale factor applied to the parenthesised argument block when \zarg is typeset in display style.
zarg-scale-text	0.9	As zarg-scale-display, in text style.
zarg-scale-script	0.63	As zarg-scale-display, in script style (*e.g.* inside an exponent).
zarg-scale-scriptscript	0.5	As zarg-scale-display, in scriptscript style.
zarg-kern	0.5mu	Horizontal kern inserted immediately before the opening parenthesis; its sign and size fine-tune the gap after the function name.

The scale factors (zarg-scale-⟨*style*⟩, see zscale) and horizontal kern (zarg-kern) options fine-tune the size and gap between a parent function identifier and its target argument block. It adds no space of its own, so any spacing around it comes from the surrounding source and math atoms alone. It is compatible with text, inline and display modes, and ensures consistent visual density across different font families. Options are governed dynamically, so changes mid-document are supported.

The following examples show a simple argument in math mode, a larger expression as function argument, and the same command used both in text and in math modes:

```
$f(x) \ne f\zarg{x}$    % comparison against parentheses
\nl
\[
  f\left( x^2 - \frac{1}{x} + 1 \right) % dynamic delim.
  \ne
  f\zarg{x^2 - \frac{1}{x} + 1}
\]
\nl
sin\zarg{\theta/2} $ = \text{sin}\zarg{\theta/2}$
```

$$f(x) \neq f(x)$$

$$f\left(x^2 - \frac{1}{x} + 1\right) \neq f\left(x^2 - \frac{1}{x} + 1\right)$$

$$\sin(\theta/2) = \sin(\theta/2)$$

The scaling follows the math style of the call, so the argument shrinks inside an exponent, and a larger kern opens the gap after the function name:

```
$e^{f(x)} \ne e^{f\zarg{x}}$    % context aware
\nl[2]
{
  \setkeys{shorthand}{zarg-kern=3mu}
  $f\zarg{x}$                   % wider gap
}
```

$$e^{f(x)} \neq e^{f(x)}$$

$$f\,(x)$$

2.7 \zdiff — differential or derivative

\zdiff The \zdiff utility typesets either a differential or a derivative operator with respect to ⟨*var*⟩, optionally raised to a given ⟨*order*⟩ (default 1). It is compatible with text, inline and display modes, wrapping its output in \ensuremath. Its options are listed in Table 5.

```
\zdiff+**[⟨order⟩]{⟨var⟩}
```

Starring:

- **none (-)** — differential, $\mathrm{d}^{\langle order \rangle}\langle var \rangle$, set as a \mathinner atom: TEX itself supplies the usual thin space against neighbouring ordinary atoms (*e.g.* right after \zarg) in display and text style, none in script styles, and the regular binary or relation spacing after operators such as \zdot.
- **one (*)** — derivative, as a standard \frac, followed by a larger space (zdiff-kern).
- **two (**)** — derivative, as an oblique \nicefrac.

Table 5: Options for \zdiff.

Option	Default	Description
zdiff-symbol	d	Symbol printed in the differential and in both derivative layouts (*e.g.* \partial, as in \zdell below).
zdiff-math-separation	10mu	Extra kern inserted before "d" in differential form when \zdiff is called with a leading + in display math style. It is *added* to the thin space that TEX already applies.
zdiff-text-separation	5mu	Same as zdiff-math-separation, applied automatically instead whenever \zdiff is typeset in text, script, or scriptscript style.
zdiff-kern	2mu	Kern inserted after the fraction in derivative form, separating it from the following operand.

The following example prints a differential, a higher-order differential, and the two derivative layouts:

```
\zdiff{x}                              % differential
\nl
\zdiff[2]{x}                           % differential, order 2
\nl
$\zdiff*{x}f \ne \zdiff*[2]{x}f$       % derivative, \frac
\nl
$\zdiff**{x}(f) \ne \zdiff**[2]{x}(f)$ % derivative, \nicefrac
```

$\mathrm{d}x$

d^2x

$\frac{\mathrm{d}}{\mathrm{d}x} f \neq \frac{\mathrm{d}^2}{\mathrm{d}x^2} f$

$\mathrm{d}/\mathrm{d}x\,(f) \neq \mathrm{d}^2/\mathrm{d}x^2\,(f)$

Leaving the bare form free of extra kerns also keeps it usable as an operand. The following example shows that a hand-built derivative of two differentials reproduces the same spacing as writing the fraction out by hand:

```
\[
\frac{\zdiff{f}}{\zdiff{x}}
=
\frac{\text{d}f}{\text{d}x}
\]
```

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}x}$$

A leading +, meaningful only in differential form, adds a kern before the "d" on top of the spacing TEX already provides (zdiff-math-separation in display style, zdiff-text-separation in text/script/scriptstyle; chosen automatically to match the ambient math style, similarly to \zint's math/text split). Intended to be used when clear visual separation is needed. The following example compares the default and increased spacing in potential use cases:

```
\begin{gather*}
  \int \vec{u} \zdot \zdiff{\vec{A}}    % default
  \quad \ne \quad
  \int f\zarg{\vec{x}} \zdiff+[2]{x}    % spaced
\end{gather*}
```

$$\int \vec{u} \cdot \mathrm{d}\vec{A} \quad \neq \quad \int f(\vec{x})\ \mathrm{d}^2x$$

`\zdell` For partial differentials and derivatives, the package provides the variant `\zdell`, which takes the same arguments, stars and leading `+` as `\zdiff` and prints ∂ instead of d.[1]

```
\zdell+**[⟨order⟩]{⟨var⟩}
```

The following example prints a partial differential, a higher-order one, and the two partial-derivative layouts:

```
$\zdell{x} \ne \zdell[2]{x}$              % partial differential
\nl
$\zdell*{x}f \ne \zdell*[2]{x}f$          % derivative, \frac
\nl
$\zdell**{x}(f) \ne \zdell**[2]{x}(f)$    % derivative, \nicefrac
```

$$\partial x \neq \partial^2 x$$

$$\frac{\partial}{\partial x} f \neq \frac{\partial^2}{\partial x^2} f$$

$$\partial/\partial x\,(f) \neq \partial^2/\partial x^2\,(f)$$

[1] It is a predefined alias of `\zdiff`; see Section 3.3.

2.8 \zcond — bar to represent evaluation conditions

\zcond The \zcond utility typesets a vertical bar to represent evaluation conditions, with ⟨*condition*⟩ as a subscript. When ⟨*expr*⟩ is supplied and no stars are given, the bar auto-sizes via \left. ⟨*expr*⟩ \right|, tracking ⟨*expr*⟩'s height. Each star instead selects a fixed bar size, in which case ⟨*expr*⟩ is printed as-is, without \left/\right wrapping. It is compatible with text, inline and display modes, wrapping its output in \ensuremath. Its options are listed in Table 6.

```
\zcond****[⟨expr⟩]{⟨condition⟩}
```

Starring:

- **none (-)** — auto-sized bar.
- **one (*)** — fixed \big|.
- **two (**)** — fixed \Big|.
- **three (***)** — fixed \bigg|.
- **four (****)** — fixed \Bigg|.

Table 6: Options for \zcond.

Option	Default	Description
zcond-scale-absolute	true	As zscale-absolute, for the condition.
zcond-scale-display	0.7	Factor by which the condition is scaled according to an unscaled subscript, for a bar in display style.
zcond-scale-text	0.7	As zcond-scale-display, in text style.
zcond-scale-script	0.6	As zcond-scale-display, in script style.
zcond-scale-scriptscript	0.6	As zcond-scale-display, in scriptscript style.
zcond-scale-star	false	false: only the condition of the auto-sized bar is scaled. true: the condition of the fixed-size bars is scaled too (otherwise it keeps the size TEX gives a subscript).

The ⟨*condition*⟩ subscript is scaled by zcond-scale-⟨*style*⟩, where ⟨*style*⟩ is the math style of the bar (see zscale). By default, only the auto-sized bar scales its condition; the fixed-size bars keep the size TEX gives a subscript. The following example shows the bar after a derivative, with an auto-sized bar around ⟨*expr*⟩, and with a fixed-size bar:

```
$\zdiff**{x}g \zcond{x=0}$          % default size
\nl
$\zcond[\zdiff*{x}g]{x=0}$          % auto-sized
\nl
$\zcond**[\zdiff*{x}g]{x=0}$        % fixed \Big|
```

$\mathrm{d}/\mathrm{d}x\, g|_{x=0}$

$\frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{x=0}$

$\frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{x=0}$

Setting zcond-scale-star to true extends the scaling to the conditions of all fixed-size bars, so that every bar size is governed by one set of factors:

```
\newcommand{\auto}{\text{auto}}
\newcommand{\fixed}{\text{fixed}}
\nl[3]
$\zcond[\zdiff*{x}g]{\auto} \ne \zcond**[\zdiff*{x}g]{\fixed}$
\nl[3.5]
{
  \setkeys{shorthand}{zcond-scale-star=true}
  \setkeys{shorthand}{zcond-scale-text=1}
  $\zcond[\zdiff*{x}g]{\auto} \ne \zcond**[\zdiff*{x}g]{\fixed}$
  \setkeys{shorthand}{zcond-scale-display=1}
  \[
    \zcond[\zdiff*{x}g]{\auto} \ne \zcond**[\zdiff*{x}g]{\fixed}
  \]
}
```

$\frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{auto}} \neq \frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{fixed}}$

$\frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{auto}} \neq \frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{fixed}}$

$\frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{auto}} \neq \frac{\mathrm{d}}{\mathrm{d}x} g\Big|_{\mathrm{fixed}}$

2.9 \zscript — four-way sub/superscripts

\zscript The \zscript engine attaches up to four independent scripts to a central ⟨*nucleus*⟩. It simultaneously handles leading (pre-indexed) subscripts ⟨*ll*⟩ and superscripts ⟨*ul*⟩, alongside conventional trailing subscripts ⟨*lr*⟩ and superscripts ⟨*ur*⟩. Omitting any optional script completely suppresses its rendering footprint. The command is compatible with text, inline and display modes. Its options are listed in Table 7.

```
\zscript+*[⟨ll⟩]*[⟨ul⟩]{⟨nucleus⟩}*[⟨lr⟩]*[⟨ur⟩]
```

Table 7: Options for \zscript. The ⟨*mode*⟩ is `math` (display) or `text` (text, script and scriptscript).

Option	Math	Text	Description
Pre-rendering			
`zscript-adaptative`	true		`true`: scripts are offset from nucleus edges. `false`: TEX's classical placement (position, frame and reference options are ignored). A leading `+` modifier locally flips this.
`zscript-reference`	(empty)		Math expression whose edges, instead of the printed nucleus's, define the positions of all scripts (see below). Empty: the nucleus itself.
`zscript-wrapper`	`\symup`		Single-argument macro applied to every script.
Starred Scripts			
`zscript-scale-absolute`	true		As `zscale-absolute`, for starred scripts.
`zscript-scale-display`	0.6		Factor by which a starred script is scaled according to an unstarred one, for a nucleus in display style.
`zscript-scale-text`	0.6		As `zscript-scale-display`, for text style.
`zscript-scale-script`	0.55		As `zscript-scale-display`, for script style.
`zscript-scale-scriptscript`	0.5		As `zscript-scale-display`, for scriptscript style.
Frame			
`zscript-`⟨*mode*⟩`-before`	0	0	Extra width reserved before the output.
`zscript-`⟨*mode*⟩`-after`	0.05	0.05	Extra width reserved after the output.
`zscript-`⟨*mode*⟩`-above`	0.2	0.2	Extra height reserved above the output.
`zscript-`⟨*mode*⟩`-below`	0.1	0.1	Extra depth reserved below the output.
Trailing Scripts			
`zscript-`⟨*mode*⟩`-ur-v`	-0.3	-0.3	Reference top edge to upper-right script bottom edge.
`zscript-`⟨*mode*⟩`-ur-h`	0.05	0.05	Reference right edge to upper-right script left edge.
`zscript-`⟨*mode*⟩`-lr-v`	0.3	0.3	Reference bottom edge to lower-right script top edge.
`zscript-`⟨*mode*⟩`-lr-h`	0	0	Reference right edge to lower-right script left edge.
Leading Scripts			
`zscript-`⟨*mode*⟩`-ul-v`	-0.3	-0.3	Reference top edge to upper-left script bottom edge.
`zscript-`⟨*mode*⟩`-ul-h`	0.05	0.05	Reference left edge to upper-left script right edge.
`zscript-`⟨*mode*⟩`-ll-v`	0.3	0.3	Reference bottom edge to lower-left script top edge.
`zscript-`⟨*mode*⟩`-ll-h`	-0.05	-0.05	Reference left edge to lower-left script right edge.

Script placement is determined by the `zscript-adaptative` option, to either use the standard TEX engine or a dedicated internal engine with high customizability. The employment of a `+` modifier directly following \zscript flips this placement choice. For both engines, applying a `*` modifier immediately before any script argument scales that script by the `zscript-scale-`⟨*style*⟩ option, where ⟨*style*⟩ is the math style of the nucleus (using `zscale`, so nesting cannot bring a size below `zscript-scale-scriptscript`). When using the dedicated engine, scripts are explicitly positioned boxes rather than using TEX's script machinery, so they may freely overlap the nucleus (Figure 2). All positions use one Cartesian system (x positive right, y positive up) and are measured from a *reference edge* (the edge of the nucleus, unless `zscript-reference` is set) to the *nearest script edge* as dimensionless ratios. Vertical options (`v`, `above`, `below`) are relative to the x-height, and horizontal ones (`h`, `before`, `after`) to the quad, both in the nucleus's math style. So, the same settings work for nuclei and scripts of any size, and rescaling a script

moves its outer edge away from the nucleus without changing its anchor. The output is a single box whose height and depth encompass the nucleus and all scripts, preventing collisions with adjacent lines and exposing its true dimensions to subsequent commands (*e.g.* further scripts or `\left`/`\right`, as shown below). The `before`, `after`, `above` and `below` options add (or remove, when negative) clearance around it.

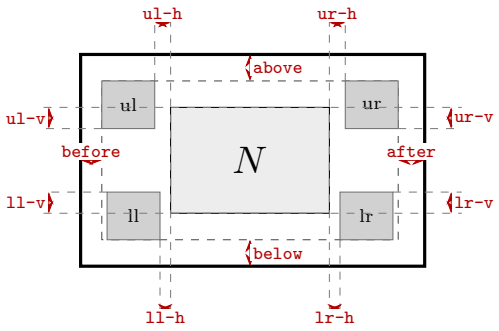


Figure 2: Schematic of the `\zscript` options, written without the `zscript-`⟨*mode*⟩`-` prefix. Each script is anchored by its nearest edge to an edge of the nucleus (dashed references), at a distance given as a ratio of the x-height (vertical) or quad (horizontal) of the style of the nucleus N. The thick frame is the box `\zscript` hands to TEX; the dashed rectangle is the union of nucleus and scripts, extended by the frame options. Distances are exaggerated for legibility.

The following examples show scripts in text and math modes, with and without scaling, inside an exponent, and demonstrate its spacing:

```
$\zscript[I][K]{M}[J][L]$                % Default scripts
\nl
$\zscript*[I]*[K]{M}*[J]*[L]$            % Scaled scripts
\nl
$e^{-\zscript{x}[J]} \ne e^{-\zscript{x}*[J]}$
\nl
\[
  \left( \zscript{\sum}[a][b] \right)    % Output size
\]
\nl
\zscript{A}[i][2]                        % Safe in text
\nl
Previous line, no overlap.\\
Tall inline output: \zscript[I][K]{\sum}[J][L]. \\
Subsequent line, no overlap.
```

$${}^{K}_{I}M^{L}_{J}$$

${}^{K}_{I}M^{L}_{J}$

$$e^{-x_J} \ne e^{-x_J}$$

$$\left(\sum_{a}^{b}\right)$$

$$A^{2}_{i}$$

Previous line, no overlap.
Tall inline output: ${}^{K}_{I}\sum^{L}_{J}$.
Subsequent line, no overlap.

Every script is wrapped in `zscript-wrapper`, which may be any macro taking one argument (*e.g.* `\textit`). This generally allows scripts to be descriptive labels instead of other variables (see below). But the nucleus is never wrapped, not even when the call sits inside a script of another `\zscript` call. This is because a nested `\zscript` call generally still stands for a(nother) variable. So, its nucleus is set back in the default math alphabet instead of inheriting the wrapper of the outer script.

```
\nl[3]
\[
  \zscript{k}[v]                    % label "v", most common use
  \ne
  \zscript{k}[\zscript{v}[b]]       % variable "v_b", also common
  \ne
  \zscript{k}[\zscript{\text{v}}[b]]  % label "v_b", uncommon
\]
```

$$k_{\mathrm{v}} \ne k_{v_{\mathrm{b}}} \ne k_{\mathrm{v_b}}$$

Three settings for the wrapper option are recommended; they differ in usage procedure for each type of script (variables, math symbols, labels, un/signed numbers):

- `\symup` (default, compact) — math symbols, digits and signs are allowed and do not require `$…$` wrapping, since the script is already in math mode. Variables must be written as `\symit{`*⟨variable⟩*`}` to not show as text. Some labels must be written as `\text{`*⟨label⟩*`}` (since, without it, *e.g.* spaces are ignored, hyphens become minus signs, *etc.*). This is the default because descriptive labels tend to be the most common type of script.
- `\text` (standard) — all labels are allowed (with spaces, hyphens, *etc.*). Variables, math symbols and signs must be wrapped by `$…$`, exactly as inside `\text`. This may become cumbersome when scripts tend to show math symbols and signs (*e.g.* exponents).
- empty (plain) — scripts are set in the ambient math alphabet, as plain TEX does, for full control. All labels must be written `\text{`*⟨label⟩*`}`.

The following examples demonstrate the outputs for each recommendation, and comment on what is unwanted in each application:

```
\nl[3]
\[
  \zscript[\mu][\symit{p}]{R}[pre\text{-}ion][-1]
  \ne
  \zscript[\mu][p]{R}[pre-ion][-1]        % variable, hyphen
\]
\nl[3]
\[
  \setkeys{shorthand}{zscript-wrapper=\text}
  \zscript[$\mu$][$p$]{R}[pre-ion][$-1$]
  \ne
  \zscript[\mu][p]{R}[pre-ion][-1]        % greek, variable, minus
\]
\nl[3]
\[
  \setkeys{shorthand}{zscript-wrapper=}
  \zscript[\mu][p]{R}[\text{pre-ion}][-1]
  \ne
  \zscript[\mu][p]{R}[pre-ion][-1]        % label
\]
```

$$ {}^{p}_{\mu}R^{-1}_{\text{pre-ion}} \neq {}^{\text{p}}_{\mu}R^{-1}_{\text{pre−ion}} $$

$$ {}^{p}_{\mu}R^{-1}_{\text{pre-ion}} \neq {}^{\text{p}}R^{\text{-1}}_{\text{pre-ion}} $$

$$ {}^{p}_{\mu}R^{-1}_{\text{pre-ion}} \neq {}^{p}_{\mu}R^{-1}_{pre-ion} $$

It should also be emphasized that the placement of scripts with the dedicated engine adapts to the size of the nucleus. For example, a script follows the nucleus's height (since positions are measured from its edges), whereas the TEX engine applies a font-dependent shift from the baseline (which is enforced even for symbols like `\int` and `\sum`). The following examples compare the placement for nuclei of different shapes with both engines:

```
\begin{gather*}
  \zscript{k}[T] \ne k_{\text{T}} = \zscript+{k}[T]
  \nl[0.5]
  \zscript{g}[T] \ne g_{\text{T}} = \zscript+{g}[T]
  \nl[0.5]
  \zscript{\sum}[T] \ne \sum_{\text{T}} \ne \zscript+{\sum}[T]
\end{gather*}
\nl[2]
{
  \setkeys{shorthand}{zscript-adaptative=false}
  \begin{gather*}
    \zscript{k}[T] = k_{\text{T}} \ne \zscript+{k}[T]
    \nl[0.5]
    \zscript{g}[T] = g_{\text{T}} \ne \zscript+{g}[T]
    \nl[0.5]
    \zscript{\sum}[T] \ne \sum_{\text{T}} \ne \zscript+{\sum}[T]
  \end{gather*}
}
```

$$ k_{\text{T}} \neq k_{\text{T}} = k_{\text{T}} $$

$$ g_{\text{T}} \neq g_{\text{T}} = g_{\text{T}} $$

$$ \sum\nolimits_{\text{T}} \neq \sum_{\text{T}} \neq \sum\nolimits_{\text{T}} $$

$$ k_{\text{T}} = k_{\text{T}} \neq k_{\text{T}} $$

$$ g_{\text{T}} = g_{\text{T}} \neq g_{\text{T}} $$

$$ \sum\nolimits_{\text{T}} \neq \sum_{\text{T}} \neq \sum\nolimits_{\text{T}} $$

Since positions follow the edges of the nucleus, a decorated nucleus such as $\tilde{\Psi}$ takes its upper scripts

higher than the bare Ψ does. The option `zscript-reference` separates the two roles: the *reference* is the expression whose edges define the positions of all scripts, while the *printed* nucleus (the argument) is what is typeset, centred on the reference. The reference is set in the style of the nucleus, so it may be any math expression. The frame still covers the printed nucleus, so the accent never collides with the line above. The option is ignored when `zscript-adaptative` is `false` or when the option is empty (default). The following example compares the default placement with a bare reference:

```
$ \zscript{A}[][base] \ne \zscript{\tilde{A}}[][high] $
\nl[2.5]
{
  \setkeys{shorthand}{zscript-reference=A}
  $ \zscript{A}[][base] = \zscript{\tilde{A}}[][base] $
}
```

$$A^{\text{base}} \neq \tilde{A}^{\text{high}}$$

$$A^{\text{base}} = \tilde{A}^{\text{base}}$$

2.10 \zprime — prime accents

\zprime The \zprime utility adorns a target ⟨*symbol*⟩ with a specified number [⟨*n*⟩] of prime marks (default $n = 1$). Its size automatically tracks the math style of the symbol being primed, through one scale factor per style (zprime-scale-⟨*style*⟩, see zscale), so primes shrink when the symbol already sits inside a script — *i.e.* an exponent, a subscript, and so on. This makes \zprime safe to nest inside other scripts without looking oversized, and lets \left/\right delimiters around a primed expression size correctly, since the output box covers both the symbol and the primes. Its options are listed in Table 8.

```
\zprime*[⟨n⟩][⟨v⟩][⟨h⟩]{⟨symbol⟩}
```

Starring:

- **none (-)** — superscript-style layout, primes flush to the symbol's right edge.
- **one (*)** — accent-style layout, primes centered above the symbol as a group, regardless of the number of marks.

Table 8: Options for \zprime. The ⟨*mode*⟩ is math (display) or text (text, script and scriptscript).

Option	Math	Text	Description
Cluster			
zprime-separation	-2mu		Mu-unit kern inserted between consecutive prime glyphs, scaled together with them. Fine-grained; use a few mu to get a visible change.
zprime-scale-absolute	true		As zscale-absolute, for the prime cluster.
zprime-scale-display	1.1		Scale factor applied to the prime cluster when the symbol being primed sits in display style.
zprime-scale-text	1.1		As zprime-scale-display, in text style.
zprime-scale-script	0.8		As zprime-scale-display, in script style.
zprime-scale-scriptscript	0.8		As zprime-scale-display, in scriptscript style.
Frame			
zprime-⟨*mode*⟩-before	0	0	Extra width reserved before the output.
zprime-⟨*mode*⟩-after	0	0	Extra width reserved after the output.
zprime-⟨*mode*⟩-above	0.1	0.1	Extra height reserved above the output.
zprime-⟨*mode*⟩-below	0.1	0.1	Extra depth reserved below the output.
Superscript Form			
zprime-⟨*mode*⟩-script-v	-1.3	-1.3	Symbol top edge to prime cluster bottom edge.
zprime-⟨*mode*⟩-script-h	-0.02	-0.02	Symbol right edge to prime cluster left edge.
Accent Form			
zprime-⟨*mode*⟩-accent-v	-0.8	-0.8	Symbol top edge to prime cluster bottom edge.
zprime-⟨*mode*⟩-accent-h	0.1	0.1	Shift of the cluster from its centred position.

All positions share one Cartesian system (x positive to the right, y positive upward) and are measured from the edges of the symbol to the nearest edge of the prime cluster: the vertical position is the y coordinate of the cluster's bottom edge, so negative values lower the primes into the symbol, while the horizontal position is the cluster's shift from its reference (the symbol's right edge for script, centred for accent). All values are *dimensionless ratios*, with vertical options (v, above, below) relative to the x-height and horizontal ones (h, before, after) to the quad, both in the symbol's math style, so the same values remain visually consistent across styles and scales, including exponents. The optional ⟨*v*⟩ and ⟨*h*⟩ (default: empty, *i.e.* the package setting for the chosen form and mode) *replace* the corresponding zprime-⟨*mode*⟩-script-v/h or zprime-⟨*mode*⟩-accent-v/h setting for the current call, providing local corrections especially useful for tall or wide nuclei such as \int or \sum, with or without their own limits (Figure 3); the examples below illustrate both layouts, script sizing, math-style adaptation, and use inside \left/\right delimiters:

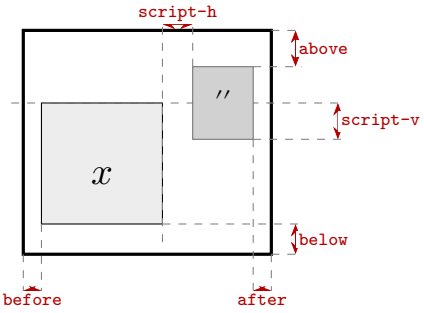
```
\zprime{x} $\ne$ \zprime*{x}                % superscript vs accent
\nl
$\zprime[2]{y} \ne \zprime*[2]{y}$          % double primes
\nl
$e^{-\zprime{x}} \ne e^{-\zprime*{x}}$      % context aware
\nl
\[
  \left(1 - \zprime{x}\right)^2             % \left and \right
  \ne
  \left(1 - \zprime*{x}\right)^2
\]
```

$$x' \neq \acute{x}$$

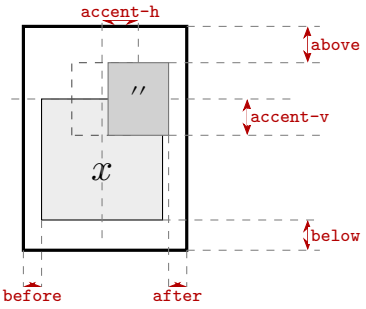
$$y'' \neq \acute{\acute{y}}$$

$$e^{-x'} \neq e^{-\acute{x}}$$

$$(1 - x')^2 \neq \left(1 - \acute{x}\right)^2$$



(a) Superscript form (unstarred). Reference: right edge.



(b) Accent form (`*`). Dashed box: centred position.

Figure 3: Schematic of the `\zprime` options, written without the `zprime-`*⟨mode⟩*`-` prefix. Each vertical option is the distance from the symbol's top edge to the cluster's bottom edge (negative values lower the primes into the symbol); each horizontal option is the shift of the cluster from its dashed reference. All are ratios of the x-height (vertical) or quad (horizontal) of the symbol's style. The thick frame is the box `\zprime` hands to TEX: it covers the *union* of the symbol and the cluster, extended by the signed `before`, `after`, `above` and `below` margins, so neighbouring lines are pushed away. The kern `zprime-separation` between consecutive primes is not drawn; distances are exaggerated for legibility.

This example shows how primes do not overlap with adjacent lines:

```
Previous line, does not overlap.\\
Tall nucleus: $\zprime*{\int} f \zdiff{x}$.\\
Subsequent line, does not overlap.
```

Previous line, does not overlap.
Tall nucleus: $\acute{\int} f \,\mathrm{d}x$.
Subsequent line, does not overlap.

The primes are scaled as a whole by the factor of the style of the symbol, so a larger factor enlarges the cluster together with the kern between its marks, which can also be tuned on its own:

```
\[
  \setkeys{shorthand}{zprime-scale-display=1.8}
  \setkeys{shorthand}{zprime-separation=-3mu}
  \zprime*[2]{\prod} \ne \zprime[2]{\prod}
\]
```

$$\acute{\acute{\prod}} \neq \prod''$$

These examples show how primes can be moved out of their default position:

```
\nl
\begin{gather*}
  \zprime*[1][][-0.06]{\int^{\infty}}      % pushed left
  \ne
  \zprime*{\int^{\infty}}
  \ne
  \zprime*{\int}^{\infty}
  \nl[3]
  \zprime*[3][][0.06]{\int^{\infty}}       % pushed right
  \ne
  \zprime*[3]{\int^{\infty}}
  \ne
  \zprime*[3]{\int}^{\infty}
  \nl[3]
  \zprime[2][-2.2][0.03]{\sum_j^{\infty}} % pushed down and right
  \ne
  \zprime{\sum_j^{\infty}}
\end{gather*}
```

$$\int^{\prime\infty} \neq \int^{\prime\infty} \neq \int^{\prime}{}^{\infty}$$

$$\int^{\prime\prime\prime\infty} \neq \int^{\prime\prime\prime\infty} \neq \int^{\prime\prime\prime}{}^{\infty}$$

$$\sum_{j}^{\infty}{}^{\prime\prime} \neq \sum_{j}^{\infty}{}^{\prime}$$

2.11 \zlim — limit with variable and value

\zlim The \zlim utility typesets the operator \lim with the script ⟨var⟩ → ⟨limit⟩ built from its two arguments. Without a leading +, the script is centred below the operator in every math style; with it, the script hangs to the lower right of the operator, like a subscript, which is the compact choice inline. One-sided limits are written in ⟨limit⟩ itself, *e.g.* 0^+. The output is a single box whose height and depth encompass the operator and its script, so adjacent lines are pushed apart. The output is wrapped in \ensuremath, so it is safe in text, inline and display mode. Its options are listed in Table 9.

```
\zlim+*{⟨var⟩}*{⟨limit⟩}
```

Starring and optional arguments:

- **none (-)** — script centred below the operator;
- **plus (+)** — script hanging to the lower right of the operator;
- a * immediately before ⟨var⟩ or ⟨limit⟩ scales that part in its normal (script) size, but additionally scales it by zlim-scale-⟨style⟩, where ⟨style⟩ is the math style of the operator (see zlim-scale-absolute); the two parts are scaled independently.

Table 9: Options for \zlim. The ⟨mode⟩ is math (display) or text (text, script and scriptscript).

Option	Math	Text	Description
Starred Parts			
zlim-scale-absolute	true		As zscale-absolute, for starred parts.
zlim-scale-display	0.6		Factor by which a starred part is scaled according to an unstarred one, for an operator in display style.
zlim-scale-text	0.6		As zlim-scale-display, in text style.
zlim-scale-script	0.43		As zlim-scale-display, in script style.
zlim-scale-scriptscript	0.43		As zlim-scale-display, in scriptscript style.
Frame			
zlim-⟨mode⟩-before	0	0	Extra width reserved before the output.
zlim-⟨mode⟩-after	0	0	Extra width reserved after the output.
zlim-⟨mode⟩-above	0	0.5	Extra height reserved above the output.
zlim-⟨mode⟩-below	0	0.5	Extra depth reserved below the output.
Below Form			
zlim-⟨mode⟩-below-v	-0.4	-0.4	Operator bottom edge to script top edge.
zlim-⟨mode⟩-below-h	0	0	Shift of the script from its centred position.
Subscript Form			
zlim-⟨mode⟩-script-v	0.3	0.3	Operator bottom edge to script top edge.
zlim-⟨mode⟩-script-h	-0.05	-0.05	Operator right edge to script left edge.

As in \zscript, the script is an explicitly positioned box rather than a product of TEX's limits machinery, so it may overlap the operator. All positions use one Cartesian system (x positive to the right, y positive upward) and are measured from the edges of the operator to the nearest edge of the script: the vertical position is the y coordinate of the script's top edge, so negative values lower it below the operator, while the horizontal position is the shift from the reference (centred for the below form, the operator's right edge for the subscript form). All values are *dimensionless ratios*, with vertical options (v, above, below) relative to the x-height and horizontal ones (h, before, after) to the quad, both in the operator's math style. A negative zlim-⟨mode⟩-after pulls the following term closer to the operator, *e.g.* a derivative.

The following example compares the two layouts, a one-sided limit, and scaled parts:

```
\begin{gather*}
  \zlim{x}{0}                          % below
  \zfrac{\sin x}{x} = 1
  \nl
  \zlim+{x}{0}                         % subscript
  \zfrac{\sin x}{x} = 1
  \nl
  \zlim{x}{0^+}                        % one-sided limit
  \zfrac{x} = +\infty
  \nl[2]
  \zlim{A}{B} f\zarg{A}
  \ne
  \zlim*{A}*{B} f\zarg{A}              % scaled
\end{gather*}
```

$$\lim_{x\to 0} \frac{\sin x}{x} = 1$$

$$\lim_{x\to 0} \frac{\sin x}{x} = 1$$

$$\lim_{x\to 0^+} \frac{1}{x} = +\infty$$

$$\lim_{A\to B} f(A) \neq \lim_{A\to B} f(A)$$

The output box keeps neighbouring lines apart in text and inline math, and the plus keeps the operator compact there; the following example shows a derivative defined as a limit, increased with a positive `zlim-math-after`:

```
Previous line, no overlap.\\
Below, inline: $\zlim{n}{\infty} a_n$.\\
Subscript, inline: $\zlim+{n}{\infty} a_n$.\\
Subsequent line, no overlap.
\nl[2]
{
  \setkeys{shorthand}{zlim-math-after=0.5}
  \[
    \zdiff*{x}f =
    \zlim{h}{0} \zfrac{f\zarg{x+h} - f\zarg{x}}{h}
  \]
}
```

Previous line, no overlap.
Below, inline: $\lim_{n\to\infty} a_n$.
Subscript, inline: $\lim_{n\to\infty} a_n$.
Subsequent line, no overlap.

$$\frac{\mathrm{d}}{\mathrm{d}x} f = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

2.12 \zint — configurable integral

\zint The \zint engine renders a configurable integral operator: like \int itself, it takes no body argument and is simply placed immediately before the integrand. The argument ⟨type⟩ selects the glyph cluster; ⟨lower⟩ and ⟨upper⟩ are its limits, each independently omittable to suppress that limit entirely. It is compatible with text, inline and display modes. Its options are listed in Table 10.

```
\zint*[⟨lower⟩]*[⟨upper⟩]**[⟨type⟩]
```

Starring and optional arguments:

- a * immediately before ⟨lower⟩ or ⟨upper⟩ scales that limit in its normal (script) size, but additionally scales it by zint-scale-⟨style⟩, where ⟨style⟩ is the math style of the integral (see zint-scale-absolute);
- the (up to two) stars immediately before ⟨type⟩ select the *placement class*, independently of math or text mode — **none** gives slant (the default: limits lean diagonally off the glyph), **one** gives stack (limits stacked below/above the glyph, as in \sum), **two** gives right (limits stacked to the glyph's right side, the most compact option, well suited to inline use);
- ⟨type⟩ (default 1) selects the glyph cluster:
  - 1 — a single $\int$;
  - 2, 3, …— n tightly-kerned integrals, for surface, volume, … integration;
  - o — a closed-path $\oint$;
  - anything else (*e.g.* n) — a bridged $\int\cdots\int$, for a symbolic order.

The most common variations of the command are identified in the following table:

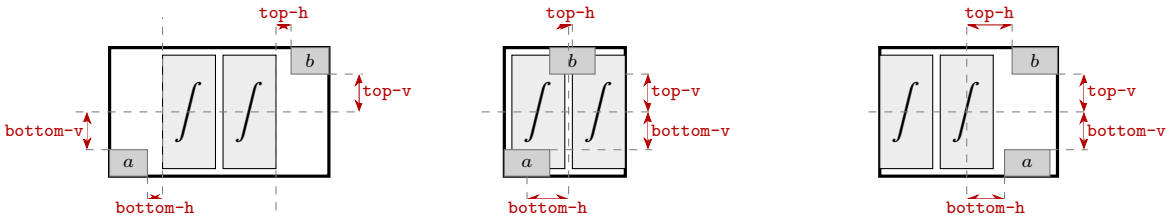
Source	Inline	Display	Meaning
\zint[a][b]	${}_a\int^b$	$\displaystyle{}_a\int^b$	single, slant limits (default)
\zint[a][b]*	$\int_a^b$	$\displaystyle\int_a^b$	single, stacked limits
\zint[a][b]**	$\int_a^b$	$\displaystyle\int_a^b$	single, right-side limits
\zint[a][b][2]	${}_a\iint^b$	$\displaystyle{}_a\iint^b$	double, slant limits
\zint[a][b]**[o]	$\oint_a^b$	$\displaystyle\oint_a^b$	closed path, right-side limits
\zint[a][b][n]	${}_a\int\cdots\int^b$	$\displaystyle{}_a\int\cdots\int^b$	bridged cluster, slant limits

Integration limits positions. Both limits are always explicitly typeset as boxes one size below the integral's own (script size in display and text style), in every mode and for every type; either may be omitted, and they may freely overlap the glyph. The complete \zint output remains a single box whose height and depth encompass the glyph and both limits, preventing collisions with adjacent lines. Positions are measured from the glyph, so one setting works for limits of any size, using a single Cartesian convention (x right, y up) whose origin is the placement class's *reference point*. For vertical measurements, the vertical centre of the glyph cluster (*i.e.* approximately the math axis), with a vertical option specifying the y of the limit's nearest edge (bottom edge of the upper limit, top edge of the lower), hence positive top and negative bottom values. For horizontal measurements, the option specifies the x of the limit's anchor, which depends on the positioning class.

Three placement classes are defined (Figures 4 and 5):

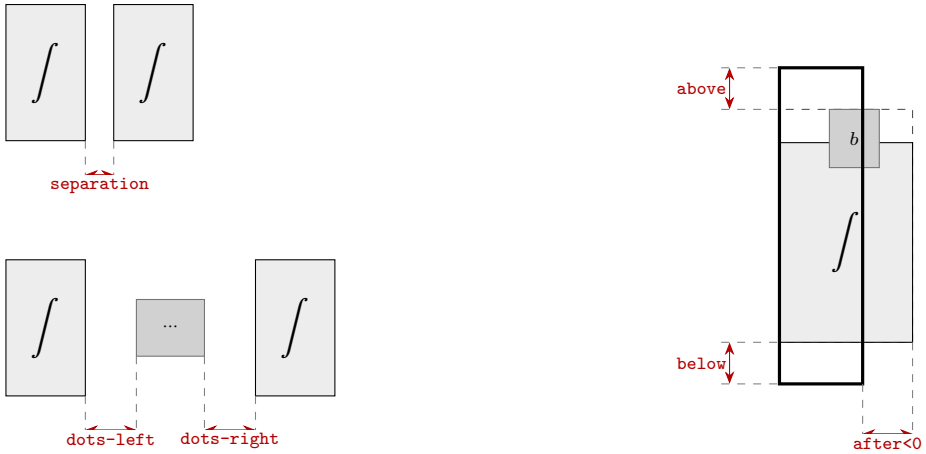
- **slant** (no star before ⟨type⟩): the reference point differs per limit, so that a cluster's limits stay fixed relative to its outer silhouette no matter how many glyphs it has — the lower limit's right edge is anchored relative to the cluster's left edge, and the upper limit's left edge is anchored relative to the cluster's right edge;

- **stack** (one star before ⟨*type*⟩): the reference is the horizontal centre of the whole cluster and the centre of each limit is anchored, so a single value suits narrow and wide limits alike;
- **right** (two stars before ⟨*type*⟩): the reference is the centre of the rightmost glyph (the glyph centre for a single integral, so it does not depend on the number of integrals), and the left edge of both limits is anchored, so wider limits grow away from the glyph.



(a) `slant` (b) `stack` (c) `right`

Figure 4: Schematic of the limit placement options for each class. A two-glyph cluster is shown throughout to make each class's reference line explicit. The vertical reference line is the cluster center for every class. The horizontal reference lines are: (**a**) the rightmost/leftmost edges of the cluster for `slant`; (**b**) the center of the cluster for `stack`; and (**c**) the center of the rightmost glyph for `right`. The thick black frame represents the output box of `\zint`. The light shade backgrounds represent glyph boxes, while dark shade represents the limits boxes. Distances in red indicate main package options without the prefix `zint-`⟨*mode*⟩`-`⟨*class*⟩`-` (⟨*mode*⟩ is `math` or `text`); distances are exaggerated for legibility.



(a) Cluster kerns (b) Output box

Figure 5: Schematic of the output of `\zint` valid for any class: (**a**) cluster kerns: `separation` between consecutive glyphs, and `dots-left` and `dots-right` on the left and right of the interior `\cdots`; (**b**) thick frame represents the box that `\zint` hands to TEX. The dashed rectangle is the union of glyph and limits, `above` and `below` extend it vertically and (a negative) `after` shifts the frame's right edge (inward) from the union's own right edge. Distances in red indicate main package options without the prefix `zint-`⟨*mode*⟩`-`; distances are exaggerated for legibility.

Between consecutive glyphs and around the `\cdots` bridge, only the cluster kerns `zint-`⟨*mode*⟩`-separation`, `-dots-left` and `-dots-right` apply; `zint-`⟨*mode*⟩`-before` and `-after` control the space before the integral and between it and the integrand, while `-above` and `-below` add or, when negative, remove clearance around the output box. All positions and frame values are dimensionless ratios of the integral's math-style x-height (vertical) or quad (horizontal), so they scale automatically, including in exponents; only the `separation`, `dots-left` and `dots-right` kerns remain lengths. The star scaling options `zint-scale-`⟨*style*⟩ are independent of mode, class and limit: ⟨*style*⟩ is `display`, `text`, `script` or `scriptscript`, identifying the integral's style, and the same four factors apply to both limits; `zint-scale-absolute` determines whether they multiply the unstarred limit size (`false`) or the normal size (`true`), as with `zscale-absolute`.

Table 10: Options for `\zint`. The ⟨*mode*⟩ is `math` (display) or `text` (text, script and scriptscript).

Option	Math	Text	Description
Starred Limits			
`zint-scale-absolute`	true		As `zscale-absolute`, for starred limits.
`zint-scale-display`	0.6		Factor by which a starred limit is scaled according to an unstarred one, for an integral in display style.
`zint-scale-text`	0.6		As `zint-scale-display`, in text style.
`zint-scale-script`	0.43		As `zint-scale-display`, in script style.
`zint-scale-scriptscript`	0.43		As `zint-scale-display`, in scriptscript style.
Cluster and Frame			
`zint-`⟨*mode*⟩`-separation`	-20mu	-10mu	Kern between consecutive `\intop` glyphs.
`zint-`⟨*mode*⟩`-dots-left`	-20mu	-10mu	Kern before the interior `\cdots` bridge.
`zint-`⟨*mode*⟩`-dots-right`	-10mu	-6mu	Kern after the interior `\cdots` bridge.
`zint-`⟨*mode*⟩`-before`	0	0	Extra width reserved before the whole output.
`zint-`⟨*mode*⟩`-after`	-0.2	-0.2	Extra width reserved after the whole output.
`zint-`⟨*mode*⟩`-above`	0.15	0.15	Extra height reserved above the output.
`zint-`⟨*mode*⟩`-below`	0.15	0.15	Extra depth reserved below the output.
Slant Limits			
`zint-`⟨*mode*⟩`-slant-top-v`	1.8	0.7	Cluster centre to upper limit bottom edge.
`zint-`⟨*mode*⟩`-slant-top-h`	-0.5	-0.1	Cluster right edge to upper limit left edge.
`zint-`⟨*mode*⟩`-slant-bottom-v`	-2	-0.7	Cluster centre to the top edge of the lower limit.
`zint-`⟨*mode*⟩`-slant-bottom-h`	-0.1	-0.1	Cluster left edge to lower limit right edge.
Right Limits			
`zint-`⟨*mode*⟩`-right-top-v`	1.4	0.2	Glyph centre to upper limit bottom edge.
`zint-`⟨*mode*⟩`-right-top-h`	0.3	0.3	Centre of rightmost glyph to upper limit left edge.
`zint-`⟨*mode*⟩`-right-bottom-v`	-1.8	-0.8	Glyph centre to lower limit top edge.
`zint-`⟨*mode*⟩`-right-bottom-h`	-0.35	0	Centre of rightmost glyph to lower limit left edge.
Stacked Limits			
`zint-`⟨*mode*⟩`-stack-top-v`	3	1.6	Glyph centre to upper limit bottom edge.
`zint-`⟨*mode*⟩`-stack-top-h`	0.3	0.2	Cluster centre to upper limit centre.
`zint-`⟨*mode*⟩`-stack-bottom-v`	-3	-1.6	Glyph centre to lower limit top edge.
`zint-`⟨*mode*⟩`-stack-bottom-h`	-0.8	-0.4	Cluster centre to lower limit centre.

The following examples compare default and scaled limits, cluster and closed-path integrals with a single lower limit, and show a vectorial integral:

```
\begin{gather*}
  \zint[a][b] f \zarg{x} \zdiff+{x}      % default
  \ne
  \zint*[a]*[b] f \zarg{x} \zdiff+{x}    % scaled
  \nl[1.5]
  \zint[A][B] f \zarg{x} \zdiff+{x}      % default
  \ne
  \zint*[A]*[B] f \zarg{x} \zdiff+{x}    % scaled
  \nl[1.5]
  \zint[\mathcal{S}][][2] f \zdiff+{A}   % cluster
  \ne
  \zint[\partial\mathcal{V}][][o]        % closed path
  \zvec{u} \zdot \zdiff{\zvec{A}}
  \nl[1.5]
  \zint[-\infty][\infty][n]              % n-dimensional
  f\zarg{\zvec{x}} \zdiff+{\zvec{x}}
\end{gather*}
```

$$\int_a^b f(x)\ \mathrm{d}x \neq \int_a^b f(x)\ \mathrm{d}x$$

$$\int_A^B f(x)\ \mathrm{d}x \neq \int_A^B f(x)\ \mathrm{d}x$$

$$\iint_{\mathcal{S}} f\ \mathrm{d}A \neq \oint_{\partial\mathcal{V}} \underline{u} \cdot \mathrm{d}\underline{A}$$

$$\int_{-\infty}^{\infty} \cdots \int f(\underline{x})\ \mathrm{d}\underline{x}$$

With a single star before ⟨*type*⟩, the `stack` class is applied. Since the limits are anchored by their own centre, narrow and wide limits such as $\mathcal{S}$ and $\partial\mathcal{V}$ share the same defaults, and a correction is only needed for exceptional cases. The following example shows stacked limits, a stacked cluster, and a comparison between the default stacked closed-path integral and a nudged version:

```
\begin{gather*}
  \zint[a][b]*
  f\zarg{x} \zdiff{x}                                % stacked limits
  \nl
  %
  \zint[\mathcal{S}][]*[2]
  f \zdiff{A}                                        % stacked cluster
  \nl[1.5]
  %
  \zint[\lim\limits_{V\to 0}\partial{V}][]*[o]
  \zvec{u} \zdot \zdiff{\zvec{A}}                    % stacked path
  \nl
  {
  \setkeys{shorthand}{zint-math-stack-bottom-h=-1}
  \zint[\lim\limits_{V\to 0}\partial{V}][]*[o]
  \zvec{u} \zdot \zdiff{\zvec{A}}                    % shifted limit
  }
\end{gather*}
```

$$\int_{a}^{b} f(x)\,\mathrm{d}x$$

$$\iint_{\mathcal{S}} f\,\mathrm{d}A$$

$$\oint_{\lim\limits_{V\to 0}\partial V} \underline{u}\cdot\mathrm{d}\underline{A}$$

$$\oint_{\lim\limits_{V\to 0}\partial V} \underline{u}\cdot\mathrm{d}\underline{A}$$

The default vertical distances suit typical limit heights; a visually taller limit — a fraction, a stacked symbol, a large integrand — may need extra clearance. The following example shows a compact expression and the same one with the upper limit raised and the integrand pulled closer:

```
\begin{gather*}
  % more compact expression
  \zint[0][{+\infty}] \sum_x \zfrac{x} \zdiff+{x}
  \nl
  {
  \setkeys{shorthand}{zint-math-after=-0.7}
  \setkeys{shorthand}{zint-math-slant-top-v=2.5}
  \zint[0][{+\infty}] \sum_x \zfrac{x} \zdiff+{x}
  }
\end{gather*}
```

$$\int_{0}^{+\infty}\sum_{x}\frac{1}{x}\,\mathrm{d}x$$

$$\int_{0}^{+\infty}\sum_{x}\frac{1}{x}\,\mathrm{d}x$$

With two stars before ⟨*type*⟩, the `right` class is applied for a more compact look. The output box includes the limits, so they clear adjacent text lines instead of colliding with them; decreasing a vertical distance — even below zero — pulls the limit toward the glyph, increasing it pushes the limit further away, without ever affecting the line spacing protection. The following examples show inline right-side limits, inline stacked limits between two regular lines, the same stacked limits with a larger vertical distance, and a comparison between inline clusters using the default and an increased spacing:

```
Inline right-side limits:
$\zint[a][b]** f\zarg{x} \zdiff{x}$.
\nl[2]
Previous line does not overlap.\\
Inline stacked limits:
$\zint[a][b]* f\zarg{x} \zdiff{x}$.\\
Subsequent line does not overlap.
\nl[4]
{
  \setkeys{shorthand}{zint-text-stack-top-v=2.2}
  \setkeys{shorthand}{zint-text-stack-bottom-v=-2.2}
  Previous line does not overlap.\\
  Increased spacing:
  $\zint[a][b]* f\zarg{x} \zdiff{x}$. \\
  Subsequent line does not overlap.
}
\nl[2]
Inline cluster: $\zint[][][2] f \zdiff{A}$.
\nl[3]
{
  \setkeys{shorthand}{zint-text-separation=-5mu}
  Increased spacing: $\zint[][][2] f \zdiff{A}$.
}
```

Inline right-side limits: $\int_a^b f(x)\,\mathrm{d}x$.

Previous line does not overlap.
Inline stacked limits: $\int\limits_a^b f(x)\,\mathrm{d}x$.
Subsequent line does not overlap.

Previous line does not overlap.
Increased spacing: $\int\limits_a^b f(x)\,\mathrm{d}x$.
Subsequent line does not overlap.

Inline cluster: $\iint f\,\mathrm{d}A$.

Increased spacing: $\iint f\,\mathrm{d}A$.

3 Package options

All tunable values of the package are options. They can be set globally, for the next call of a command only, or in reusable variants of a command, as described in the following subsections.

3.1 Global options

All tunable constants are exposed as key–value options via a unified interface. All keys belong to the family `shorthand`, so that is the first argument of `\setkeys` (not the package name). Every option is a plain macro read at call time, so it may be given to `\usepackage`, changed at any point with `\setkeys{shorthand}{key=value}`, or overridden for a single call by doing so inside a TEX group. The table that follows the syntax of each command lists its options, default values and their effect(s); captions explain eventual placeholders.

3.2 Local options

`\zlocal` The `\zlocal` utility defines a prefix command that changes options for the next call of a package command only. It accepts arguments at call time, so a single definition can take a different value at each call.

```
\zlocal{⟨new⟩}{⟨commands⟩}{⟨key=value,…⟩}
```

The list *⟨commands⟩* contains names of package commands that have options, with (*e.g.* `\zscript`, `\zlim`) or without backslash (*e.g.* `zscript`, `zlim`). If *⟨commands⟩* is empty, all commands are used. The *⟨key⟩* is automatically prepended with the option prefix of each listed command, so a key that the commands share (*e.g.* the frame options) changes all of them; a key that is missing for some commands is skipped for them. Every key must exist in at least one command, which is validated during definition. The values may contain `#1`, …, `#9`, which become the arguments of *⟨new⟩*, as many as the highest one used. An existing macro (including the package commands) cannot be overwritten, but *⟨new⟩* can be redefined. A prefix must be placed directly before the command it changes. If it is followed by anything else, it is discarded, so it can never leak into a later call. Consecutive prefixes add up, and the last one wins for an option set twice. Commands of the same kind typeset inside the arguments inherit the values.

For convenience, the package predefines some prefixes, as shown below.

`\zbefore` `\zafter` The prefixes `\zbefore` and `\zafter` set the options for common adjustments, the space before or after the output of `\zscript`, `\zprime`, `\zlim` and `\zint`. They are equivalent to `\zlocal{\zbefore}{zscript, zprime, zlim, zint}{math-before=#1, text-before=#1}` and `\zlocal{\zafter}{zscript, zprime, zlim, zint}{math-after=#1, text-after=#1}`, and, as any prefix, can be redefined.

```
\zbefore{⟨ratio⟩}     \zafter{⟨ratio⟩}
```

The following examples push an integrand farther from the integral, and give a pre-indexed nuclide room from its coefficient:

```
\nl[2]
\begin{gather*}
  \zint[0][{+\infty}] \zfrac{x} \zdiff+{x}
  \ne
  \zafter{0.3}\zint[0][{+\infty}] \zfrac{x} \zdiff+{x}
  \nl
  2\zscript[4][2]{\mathrm{He}}
  \ne
  2\zbefore{0.3}\zscript[4][2]{\mathrm{He}}
\end{gather*}
```

$$\int_0^{+\infty} \frac{1}{x}\,\mathrm{d}x \neq \int_0^{+\infty} \frac{1}{x}\,\mathrm{d}x$$

$$2\,{}^{2}_{4}\mathrm{He} \neq 2\,{}^{2}_{4}\mathrm{He}$$

`\zref` `\zwrapper` The prefixes `\zref` and `\zwrapper` set the options `zscript-reference` and `zscript-wrapper` for the next call of `\zscript` (see Section 2.9). They are equivalent to `\zlocal{\zref}{zscript}{reference=#1}` and `\zlocal{\zwrapper}{zscript}{wrapper=#1}`, and, as any prefix, can be redefined.

```
\zref{⟨expr⟩}     \zwrapper{⟨macro⟩}
```

The following examples use `\zref` to address the same case presented in Section 2.9, `\zwrapper` to switch the wrapper of a label, and two new prefixes to demonstrate how definitions can be combined:

```
\zlocal{\zlow}{zscript}{math-lr-v=#1, text-lr-v=#1}
\zlocal{\zgap}{zscript, zlim}{math-after=#1, text-after=#1}
\nl[2.5]
\begin{gather*}
  \zscript{\tilde{A}}[][high] \ne \zref{A}\zscript{\tilde{A}}[][base]
  \nl
                        \zscript{x}[base]
  \ne          \zref{g}\zscript{x}[low]
  \ne \zlow{-1}\zref{g}\zscript{x}[lower]
  \nl
  \zscript{R}[pre-ion] \ne \zwrapper{\text}\zscript{R}[pre-ion]
  \nl[0.5]
  \zscript{x}[a]y \ne \zgap{1}\zscript{x}[a]y
  \nl[0.5]
  \zlim{n}{\infty}y \ne \zgap{1}\zlim{n}{\infty}y
\end{gather*}
```

$$\tilde{A}^{\text{high}} \neq \tilde{A}^{\text{base}}$$

$$x_{\text{base}} \neq x_{\text{low}} \neq x \atop \text{lower}$$

$$R_{\text{pre-ion}} \neq R_{\text{pre-ion}}$$

$$x_{\text{a}}y \neq x_{\text{a}}\ \ y$$

$$\lim_{n\to\infty} y \neq \lim_{n\to\infty}\ \ y$$

3.3 Aliases

`\zalias` The `\zalias` utility defines a new command based on one of the package commands that have options, but with its own option values. It can be used when different sets of options need to be re-used multiple times in the same document. The ⟨*new*⟩ command takes the same arguments of the ⟨*base*⟩ command.

`\zalias{`⟨*new*⟩`}{`⟨*base*⟩`}{`⟨*key=value,…*⟩`}`

The list ⟨*key=value,…*⟩ contains option names and values, which are validated during definition. The ⟨*key*⟩ is automatically prepended with the option prefix of the ⟨*base*⟩ family (*e.g.* `zscript-` for `\zscript`; `\zvec` and `\zmat` use `zline-`). This guards `\zalias` against changing the options of an unrelated command. Listed key-value pairs are changed inside a dedicated group, so unlisted options continue using the values defined outside the definition (usually the package defaults). An alias may be based on another alias and may be redefined at any point, but all definitions are frozen. A later redefinition of an alias does not update its sub-aliases already based on it. An existing macro, including every command of the package, cannot be overwritten.

The following examples use `\zalias` to address the same case presented in Section 2.9, as well as demonstrate possible strategies to adjust low subscripts caused by symbol underlining:

```
\zalias{\zscriptA}{\zscript}{reference=A}
\zalias{\zscriptMid}{\zscript}{math-lr-v=1, text-lr-v=1}
\zalias{\zscriptHigh}{\zscript}{math-lr-v=1.6, text-lr-v=1.6}
\nl[3.5]
\begin{gather*}
  \zscript{A}[][base] \ne \zscript{\tilde{A}}[][high]
  \nl[0.5]
  \zscript{A}[][base] =  \zscriptA{\tilde{A}}[][base]
  \nl[0.1]
  \nl[0.1]
                  \zmat{X_\text{b}}  \ne          \zmat{X}_\text{b}
  \nl[0.5]
              \zmat{\zscript{X}[b]}  \ne      \zscript{\zmat{X}}[b]
  \nl[0.5]
           \zmat{\zscriptMid{X}[b]}  \ne   \zscriptMid{\zmat{X}}[b]
  \nl[0.5]
          \zmat{\zscriptHigh{X}[b]}  \ne  \zscriptHigh{\zmat{X}}[b]
\end{gather*}
```

$$A^{\text{base}} \neq \tilde{A}^{\text{high}}$$

$$A^{\text{base}} = \tilde{A}^{\text{base}}$$

$$\underline{\underline{X_{\text{b}}}} \neq \underline{\underline{X}}_{\text{b}}$$

$$\underline{\underline{X_{\text{b}}}} \neq \underline{\underline{X}}_{\text{b}}$$

$$\underline{\underline{X_{\text{b}}}} \neq \underline{\underline{X}}_{\text{b}}$$

$$\underline{\underline{X^{\text{b}}}} \neq \underline{\underline{X}}_{\text{b}}$$

When using `\zlocal`, the prefix values are applied inside the group of the call after the keys of an alias. So a prefix wins when alongside an alias, and can undo its option for call:

```
\zalias{\zscriptLow}{\zscript}{math-lr-v=-1}
\zlocal{\zlr}{zscript}{math-lr-v=#1}
\nl[4]
\begin{gather*}
                \zscript{x}[base]
  \ne           \zscriptLow{x}[low]
  \ne \zlr{-2}\zscriptLow{x}[lower]
  \nl[0.5]
                      \zscriptLow{x}[low]
  \ne \zlr{\zreset}\zscriptLow{x}[base]
\end{gather*}
```

$$x_{\text{base}} \neq x_{\text{low}} \neq x_{\text{lower}}$$

$$x_{\text{low}} \neq x_{\text{base}}$$

`\zreset` Since option values are applied around the whole call, they are inherited by nested calls of the same base command. The macro `\zreset` restores the value an option had when the *outermost* enclosing alias call began (or current values, if there is none). That call takes a snapshot of its command's options before applying its own keys, and nested alias calls keep it; *i.e.* the restored values follow the `\setkeys` and groups around the place where the alias is *called*, not where it was defined, at any nesting depth. A `\setkeys` inside the arguments of the call is not part of the snapshot. Alternatively, the key `reset` does the same for all options of the base command at once. Keys are applied in order, so `reset` only undoes the keys that precede it. Both also work in the keys of `\zlocal`, whose values may be `\zreset` or contain it in an argument, as the examples below show.

The following example shows that `reset` only affects the keys before it:

```
\zalias{\zscriptUndo}{\zscript}{text-lr-v=-1, reset}
\zalias{\zscriptKeep}{\zscript}{reset, text-lr-v=-1}
\nl[4]
$
  \zscript{A}[base] = \zscriptUndo{A}[base] \ne \zscriptKeep{A}[low]
$
```

$$A_{\text{base}} = A_{\text{base}} \neq A_{\text{low}}$$

and these show the inheritance and how an inner alias with `\zreset` avoids it:

```
\zalias{\zscriptLow}{\zscript}{text-lr-v=-1}
\zalias{\zscriptRaw}{\zscript}{text-lr-v=\zreset}
\nl[3]
$ {x}_{k_{b}} $
\nl
$ \zscript{x}[\zscript{k}[b]] = \zscript{x}[\zscriptRaw{k}[b]] $
\nl
$ \zscript{x}[\zscriptLow{k}[b]] \ne \zscript{x}[\zscriptRaw{k}[b]] $
\nl[0.5]
$ \zscriptLow{x}[\zscript{k}[b]] \ne \zscriptLow{x}[\zscriptRaw{k}[b]] $
\nl[0.5]
\nl[0.5]
$ \zscriptLow{x}[\zscriptLow{k}[b]] \ne \zscriptLow{x}[\zscriptRaw{k}[b]] $
```

$$x_{k_b}$$

$$x_{k_{\text{b}}} = x_{k_{\text{b}}}$$

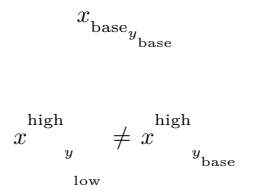
$$x_{k_{\text{b}}} \neq x_{k_{\text{b}}}$$

$$x_{k_{\text{b}}} \neq x_{k_{\text{b}}}$$

$$x_{k_{\text{b}}} \neq x_{k_{\text{b}}}$$

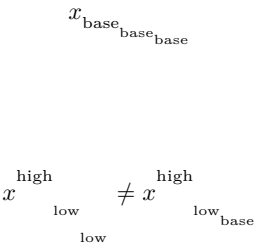
The restored value is the one acting where the outermost alias is called, not one of the intermediate aliases:

```
\zalias{\zscriptLow}{\zscript}{text-lr-v=-3}
\zalias{\zscriptHigh}{\zscript}{text-lr-v=3}
\zalias{\zscriptRaw}{\zscript}{text-lr-v=\zreset}
\newcommand{\high}{\text{high}}
\newcommand{\base}{\text{base}}
\newcommand{\low}{\text{low}}
\nl[7]
$ \zscript{x}[\zscript{\base}[\zscript{y}[base]]] $
\nl[2]
$
  \zscriptHigh{x}[\zscriptLow{\high}[\zscript{y}[low]]]
  \ne
  \zscriptHigh{x}[\zscriptLow{\high}[\zscriptRaw{y}[base]]]
$
```



and the restored value is also not the one at definition:

```
\zalias{\zscriptLow}{\zscript}{text-lr-v=-3}
\zalias{\zscriptHigh}{\zscript}{text-lr-v=3}
\zalias{\zscriptRaw}{\zscript}{text-lr-v=\zreset}
\newcommand{\high}{\text{high}}
\newcommand{\base}{\text{base}}
\newcommand{\low}{\text{low}}
\nl[7]
$ \zscript{x}[\zscript{\base}[\zscript{\base}[base]]] $
\nl[4]
{
  \setkeys{shorthand}{zscript-text-lr-v=0}
  $
    \zscriptHigh{x}[\zscriptLow{\high}[\zscript{\low}[low]]]
    \ne
    \zscriptHigh{x}[\zscriptLow{\high}[\zscriptRaw{\low}[base]]]
  $
}
```



4 Combined Examples

4.1 Fundamental theorem of calculus

```
\begin{gather*}
  \zint[a][b] f(x) \, \zdiff{x} =
  F(x)\zcond**{x=b} - F(x)\zcond**{x=a}
\end{gather*}
```

$$\int_a^b f(x)\ \mathrm{d}x = F(x)\Big|_{x=b} - F(x)\Big|_{x=a}$$

4.2 Tensor with four-way scripts and evaluation bar

```
Ricci tensor components, evaluated at the origin:
\[
  \zscript[\mu][\nu]{Ri}[\alpha][\beta]
  \zcond**[\,]{x=0}
  = \zfrac[\zscript{Ri}[\mu\nu]]{\kappa}
\]
```

Ricci tensor components, evaluated at the origin:

$$\left.{}^{\nu}_{\mu}Ri^{\beta}_{\alpha}\right|_{x=0} = \frac{(Ri_{\mu\nu})}{\kappa}$$

4.3 Divergence theorem

```
\[
  \zint[\partial\mathcal{V}][][2]
  \zvec{u} \zdot \hat{n} \zdiff+{A}
  =
  \zint[\mathcal{V}][][3]
  \nabla \zdot \zvec{u} \zdiff+{V}
\]
The flux of a vector field through a closed surface
  equals the divergence integrated over the enclosed
  volume.
```

$$\iint_{\partial\mathcal{V}} \underline{u}\cdot\hat{n}\ \mathrm{d}A = \iiint_{\mathcal{V}} \nabla\cdot\underline{u}\ \mathrm{d}V$$

The flux of a vector field through a closed surface equals the divergence integrated over the enclosed volume.

4.4 Multi-dimensional integration

```
Integration in multiple dimensions:
\begin{gather*}
  \zint[a][b] f(x) \zdiff+{x}
  \\
  \zint[\mathcal{S}][][2] f \zdiff+{A}
  \\
  \zint[V][][3] \rho \zdiff+{V}
  \\
  \zint[\mathbb{R}^n][][n] g \zdiff+[n]{x}
\end{gather*}
```

Integration in multiple dimensions:

$$\int_a^b f(x)\ \mathrm{d}x$$

$$\iint_{\mathcal{S}} f\ \mathrm{d}A$$

$$\iiint_{V} \rho\ \mathrm{d}V$$

$$\idotsint_{\mathbb{R}^n} g\ \mathrm{d}^n x$$

4.5 Gaussian integral

```
\[
  \zint[-\infty][{+\infty}]
  e^{-x^2} \, \zdiff{x} = \sqrt{\pi}
\]
```

$$\int_{-\infty}^{+\infty} e^{-x^2}\ \mathrm{d}x = \sqrt{\pi}$$

4.6 Circulation of a vector field

```
\[
  \zint[\partial\mathcal{S}][][o]
  \vec{F} \zdot \zdiff{\vec{r}}
  =
  \zint[\mathcal{S}][][2]
  \left( \vec{\nabla} \times \vec{F} \right)
  \zdot \zdiff{\vec{A}}
\]
Stokes' theorem relates the circulation around a closed
   curve to the flux of the curl through any bounded
  surface.
```

$$\oint_{\partial S} \vec{F} \cdot \mathrm{d}\vec{r} = \iint_{S} \left(\vec{\nabla} \times \vec{F}\right) \cdot \mathrm{d}\vec{A}$$

Stokes' theorem relates the circulation around a closed curve to the flux of the curl through any bounded surface.

4.7 Arc length of a parametrised curve

```
If \zprime{\zvec{r}}\zarg{s} is the tangent
vector with respect to arc length $s$ of curve
vector position \zvec{r}, then:
\begin{gather*}
  \zprime{\zvec{r}} = \zdiff*{s}\zvec{r}
  \quad \implies
  \nl
  L = \zint[0][\ell]
  \left\| \zprime{\zvec{r}}\zarg{s} \right\|
  \zdiff+{s} \zpunct{.}
\end{gather*}
```

If $\underline{r}'(s)$ is the tangent vector with respect to arc length s of curve vector position $\underline{r}$, then:

$$\underline{r}' = \frac{\mathrm{d}}{\mathrm{d}s}\underline{r} \quad \implies$$

$$L = \int_{0}^{\ell} \left\|\underline{r}'(s)\right\| \ \mathrm{d}s \quad .$$

4.8 Moment of inertia of spherical shell

```
The moment of inertia of a thin spherical shell is:
\[
  I = \zint[0][\pi][2]
  \zfrac{2}{3} m r^2 \sin^3\zarg{\theta}
  \zdiff+{\theta} \zpunct{.}
\]
```

The moment of inertia of a thin spherical shell is:

$$I = \iint_{0}^{\pi} \frac{2}{3} m r^2 \sin^3(\theta) \ \mathrm{d}\theta \quad .$$

4.9 Heat equation solution

```
Equation \ref{eq:green} applies the Green
function for the heat equation:
\begin{equation}
  \label{eq:green}
  u\zarg{x,t} =
  \zfrac{\sqrt{4\pi\kappa t}}
  \zint[-\infty][{+\infty}]
  u_0\zarg{\zprime{x}}
  \cdot e^{
      -\zfrac
      {\left( x-\zprime{x} \right)^2}
      {4\kappa t}
  }
  \zdiff+{\zprime{x}}
  \zpunct{.}
\end{equation}
```

Equation 4 applies the Green function for the heat equation:

$$u(x,t) = \frac{1}{\sqrt{4\pi\kappa t}} \int_{-\infty}^{+\infty} u_0(x') \cdot e^{-\frac{(x-x')^2}{4\kappa t}} \ \mathrm{d}x' \quad . \ (4)$$

4.10 Beer–Lambert law

```
Light crossing a medium of non-uniform density obeys the
generalised Beer--Lambert law:
\begin{gather*}
  I\zarg{L}
  = I_0 \, e^{-\zint[0][{L}]{\kappa\zarg{x}} \zdiff+{x}}
  \zpunct{.}
\end{gather*}
```

Light crossing a medium of non-uniform density obeys the generalised Beer–Lambert law:

$$I(L) = I_0 \, e^{-\int_0^{L} \kappa(x) \, \mathrm{d}x} \,.$$

5 Change History

v1.1.0 (2026/10/28)

Local prefixes and aliases of commands. New options. Standardized option declaration.

- **Warning: breaking change in a minor release.** The keys of `\zscale` were renamed from `scale-`⟨*style*⟩ and `scale-absolute` to `zscale-`⟨*style*⟩ and `zscale-absolute`. They were a typo against the rule that every key carries the full name of its command, and the version is bumped to v1.1.0 instead of v2.0.0 for that reason only, but it *does* break existing documents: replace the old keys, in `\usepackage` and in `\setkeys`; no alias is kept.
- **Scale keys.** The keys `scale-`⟨*style*⟩ used to live in a global section of the package because `\zscale` had a scaling shared by all commands. Since every command now owns its four factors and switch, that distinction was gone, and the keys were moved to the section of `\zscale`, where they belong.
- **Package options.** The subsection on options became part of a new *Package options* section, with the subsections *Global options* (moved from the user interface), *Local options* and *Aliases*.
- **Declaration of options.** All options are now declared by one generator in the global section, instead of one helper per command or by hand, with unchanged macro names and defaults. The generator also registers every option of each command, which is what `\zlocal` and `\zalias` use to take snapshots and to apply `reset`.
- `\zlocal`: new command that defines prefix commands, which change options for the next call of a command only. They take up to nine arguments, can change the options of several commands at once, add up when combined and are applied after the keys of an alias. A prefix that is not followed by a command with options is discarded.
- `\zbefore`, `\zafter`: new prefixes, predefined with `\zlocal`, that set the spacing before and after the output of `\zscript`, `\zprime`, `\zlim` and `\zint`.
- `\zref`, `\zwrapper`: new prefixes, predefined with `\zlocal`, that set `zscript-reference` and `zscript-wrapper` for the next call of `\zscript`.
- `\zalias`: new command that defines variants of any command with options, with their own option values. The key prefix of the base command is prepended automatically, so an alias cannot change other commands' options. Aliases may be based on other aliases, may be redefined, and never overwrite a command of the package. The value `\zreset` and the key `reset` restore options to the values they had when the outermost enclosing alias call began.
- `\zdell`: alias of `\zdiff`, created with the new option `zdiff-symbol`.
- `\zscript`: new option `zscript-reference`, a math expression whose edges, instead of the printed nucleus's, define the positions of all scripts, so that a decorated nucleus (*e.g.* $\tilde{\Psi}$) takes its scripts where the bare one does. The leading `+` is now resolved after the keys of an alias are applied. The nucleus of a nested `\zscript` is never wrapped.
- `\zcond`: the condition is now scaled through its own options `zcond-scale-`⟨*style*⟩, `zcond-scale-absolute` and the new switch `zcond-scale-star`, keyed to the math style of the bar. By default the output is essentially unchanged.

v1.0.0 (2026/10/14)

Package renamed to math-shorthandz. First public release.

- **Package name.** Renamed from zmath to math-shorthandz (files, `\usepackage` and repository). The final "z" keeps the name consistent with the `\z...` command prefix. Commands and option keys are unchanged.
- **Key family.** The keyval family was renamed from `zmath` to `shorthand`: replace `\setkeys{zmath}{...}` by `\setkeys{shorthand}{...}`; no alias is kept. Options given to `\usepackage` are unaffected.
- **Release numbering.** The package was intended to be released as v1.0.0 under the name zmath. Since CTAN and TEX Live required a less generic name, the first public tag was moved to this release, and the former zmath releases are recorded as v0.12.0 and v0.13.0.

v0.13.0 (2026/10/12)

Per-call control of the placement of scripts. Option rename.

- `\zscript`: the switch `zscript-adaptative` (which was missing from v0.12.0) selects between the placement from the edges of the nucleus and TEX's own placement, and a leading `+` flips it for a single call, whatever its value.
- `\zscript`: the option `zscript-font` was renamed `zscript-wrapper`, since it selects a macro applied to every script rather than a font. Replace `font` by `wrapper` in the key; its meaning and values are unchanged, and no alias is kept.

v0.12.0 (2026/10/09)

Every command was rewritten on a common footing and shared naming convention (`\z...`). New commands added. Last major rewrite under the name `zmath`, originally intended as v1.0.0.

- **All commands.** Every tunable constant is now a key–value option, read at call time and settable globally or locally with `\setkeys`, replacing fixed constants and hard-coded kerns. All commands are safe in text, inline and display mode, and adapt to the math style of the call.
- **Scaling.** Scaling follows the math style of the call through one shared mechanism with four factors per command (display, text, script, scriptscript), an absolute or relative reading of these factors, and a guard that stops nested scaled content from shrinking below scriptscript size.
- **Positions.** Distances are dimensionless ratios of the x-height and quad of the style, so one setting stays valid in exponents. Commands that place parts explicitly return a single box whose height and depth cover all of them, so neighbouring lines no longer collide.
- `\zpunct` (supersedes `\eqpunct`): works both in unstarred and starred environments through `\tag*`, and discards the punctuation on `\notag`/`\nonumber` rows so it cannot leak into a later number or `\eqref`.
- `\zfrac`: the layout is now always wrapped in `\ensuremath`, so operands may contain math even when called from text, and parentheses are always `\left` `\right` delimiters; the text-mode `\mbox` workaround is gone.
- `\zscale`: the single fixed factor became four per-style factors, with the absolute/relative switch and the nesting guard above.
- `\zdiff` (supersedes `\diff`): the differential is a `\mathinner` atom, so TEX supplies the spacing instead of a fixed `10mu` kern, which also removes double spacing after operators such as `\zdot`; a leading `+` adds extra separation, chosen by math style, and the derivative kern is an option.
- `\zscript`: the four scripts are explicitly placed boxes with per-style, per-mode ratio options instead of TEX's script machinery, so they may overlap the nucleus; a star scales a script by the style factors; and every script is wrapped in the new `zscript-font` option, by default `\symup`.
- `\zprime` (supersedes `\primed`): primes scale with the math style of the symbol, so they are safe in exponents; the accent form is placed explicitly rather than through `\accentset`, with the cluster centred as a group; the vertical and horizontal positions are options, replaceable per call, instead of a single manual `\raisebox` offset; and the output box covers symbol and primes, so `\left`/`\right` and later scripts see the true height, replacing the `\vphantom` and `\smash` workarounds.
- `\zint`: the display and inline constants were merged into options for two modes and three placement classes (`slant`, `stack`, `right`), selected by zero, one or two stars before the type; limits are always explicit boxes with independent anchors, so one geometry serves all types; and the frame (`before`, `after`, `above`, `below`) and the cluster kerns are options.
- New commands: `\zline` (with `\zvec` and `\zmat`), `\zdot`, `\zarg`, `\zcond` and `\zlim`.

v0.11.0 (2025)

Flexible fraction incorporated into the package.

- `\zfrac`: stars select `\frac`, `\nicefrac` or slash form; braces or brackets select plain or parenthesised operands; a single argument is the denominator. Math or text mode is

detected at the call site.

v0.10.0 (2022-2025)

Single macro for integrals with slant/stack limits.

- `\zint`: single, multiple, closed-path and bridged integrals, with "side" or "stack" limits (one star before the type) chosen separately for display and inline style, using named spacing constants.

v0.9.0 (2023-2025)

First consolidated package for personal use.

- Shared scale coefficient.
- `\zscale`: uniform scaling of math content.
- `\zscript` (supersedes `\varsubsup`): pre- and post-scripts, with optional scaling per script by starring (rather than suppressed scaling, as in `\varsubsup`)

v0.8.0 (2024-2025)

Single macro for sub/superscripts.

- `\varsubsup`: shorthand for trailing scripts, scaled by default.

v0.7.0 (2021-2024)

Single macro for prime marks as accent or superscripts.

- `\primed`: prime accent via `\accentset`, or `\superscript`; optional number of primes and manual height offset.

v0.6.0 (2020-2023)

Single macro for differentials and derivatives.

- `\diff`: differential or derivative, with one star (`\frac`) or two (`\nicefrac`), with an optional order and fixed kerns around the output.

v0.5.0 (2020-2022)

Equation punctuation for numbered equations. Shortcut for function arguments.

- `\eqpunct`: adds punctuation mark to numbered equations through a redefined `\tagform@` hook.
- `\arg`: automatic `\left(/\right)` wrapping of arguments for functions.

v0.0.0-v0.4.0

…

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Z

\zafter *27*
\zalias *28*
\zarg *9*
\zbefore *27*
\zcond *12*
\zdell *11*
\zdiff *10*
\zdot *8*
\zfrac *5*
\zint *22*
\zlim *20*
\zline *4*
\zlocal *27*
\zmat *4*
\zprime *17*
\zpunct *3*
\zref *27*
\zreset *29*
\zscale *6*
\zscript *13*
\zvec *4*
\zwrapper *27*