

The *intexgral* package

4.3.1

Valentin Dao*

September 21, 2026

Abstract

While integral composition is prevalent in $\LaTeX$, it often proves to be impractical. When the expression becomes complex, it is then difficult to modify its various elements in an illegible source code. To address this problem, the *intexgral* package offers a central macro whose only argument is the integrand. Everything else (symbol, limits, variables...) can be modified using a `\key=value` interface. Since this package is written in *expl3*, it depends on *l3kernel*.

*E-mail: vdao.texdev@gmail.com

Contents

Abstract	1
I Documentation	4
1 Package options	5
2 Main macro overview	5
3 List of keys	6
3.1 Integration limits	6
3.1.1 Defining the limits	6
3.1.2 Modify the style	7
3.2 Display modes	7
3.3 Symbol (and more about limits)	8
3.3.1 Selecting a symbol	8
3.3.2 Generating many integrals	8
3.3.3 Defining boundaries on an ad hoc basis	9
3.3.4 Recurring limits motif	9
3.4 Differentials	10
3.4.1 Specifying the integration variables	10
3.4.2 Including the Jacobian	10
3.4.3 Modifying the differential symbol	10
3.4.4 Creating a path integral	11
3.4.5 Adding an exponent	11
4 Additional macros	11
4.1 Swapping the position of the differentials	11
4.2 Macros contextuelles	12
4.3 Typesetting a primitive	13
4.4 Looking back at the <i>limits</i> key	14
4.4.1 Defining a keyword	14
4.4.2 Writing symmetric limits	15
4.5 Looking back at the <i>variables</i> key	16
4.5.1 Defining a keyword	16
4.6 Looking back at the <i>symbol</i> key	17
5 Special syntax	18
5.1 Presentation of the syntax	18
5.2 Functioning of the syntax	19
6 Additional parameters	20
Changelog	22
II Implementation	24
1 Initialisation	25
1.1 Package declaration	25

1.2	Preliminary verifications	25
2	Package options	26
2.1	Variable definition	26
2.2	Package option declaration	26
3	Messages	27
4	Useful variants	29
5	Internal variables	29
5.1	Tokens	29
5.2	Booleans	30
5.3	Sequences and clists	31
5.4	Property lists	31
5.5	Muskip	32
5.6	String	32
6	Auxiliary macros and sequences	32
6.1	Main sequences	33
6.2	Supplementary sequences	34
7	Integration limits	34
8	Variables	38
9	Special syntax	38
10	Typesetting the integral	41
10.1	Mode <i>default</i>	42
10.2	Mode <i>nested</i>	43
10.3	Mode <i>product</i>	45
10.4	Final typesetting	46
11	Key declaration	47
12	User-level macros	52
12.1	Keywords for limits	52
12.2	Keywords for variables	53
12.3	Keywords for symbols	54
12.4	Other macros	55
13	Package configuration	57
13.1	Symbol	57
13.2	Limits	57
13.3	Variables	57
13.4	Default parameters	57
	Index	59

License

© 2025 Valentin Dao, published under the ~~La~~TeX Project Public License (LPPL) 1.3c

Fonts

Chronicle Text G3 Roman

© 2002, 2007 Hoefler & Frere-Jones.

Whitney-Medium

© 1996, 2009 Hoefler & Frere-Jones.

Monospace Argon Medium

© 2023, GitHub

Repository

 See [GitHub repository](#).

Acknowledgements

Many thanks to Plante and Slurpy for accompanying me on this adventure of learning T_EX, your pieces of advice were priceless. I'd also like to thank Anthony, who has always kindly reviewed the new versions and given me ideas for those to come.

PART I

DOCUMENTATION

1 Package options

`\intexgralsetup` `\intexgralsetup` {<package options>}

new: v2.0.0

These options can be declared in a classic way with `\usepackage` or with `\intexgralsetup` in the preamble.

`invert-limits` `invert-limits`=<`true` | `false`>

This key inverts the limit order convention. The entirety of the document can only follow one convention.

`invert-diff` `invert-diff`=<`true` | `false`>

update: v3.0.0

In physical sciences, it is common to see the differentials placed before the integrand. This key hence swaps their position.

`limits-mode` `limits-mode`=<`limits` | `nolimits`>

new: v3.0.0

Applies `\limits` or `\nolimits` to all integrals.

`italic` `italic`=<`true` | `false`>
`upright` `upright`=<`true` | `false`>

update: v4.0.0

Applies an upright or italic “d” for the differentials.

Remark: Since the version v4.0.0, the *derivative* package is no longer used to manage differentials. The number of options is therefore more limited but should suit most use cases in the context of integrals.

2 Main macro overview

`\integral` `\integral` [<list of keys>] {<integrand>}

update: v3.0.0

This macro is used to typeset integrals. It must, of course, be used in math mode only. Here is a first example in its simplest form:

```
\begin{equation}
\integral{x}
\end{equation}
```

$$\int x \, dx \tag{1}$$

Since this macro focuses on the use of keys, the first part of this documentation will present them, grouping them by domain. The second part will introduce a few

additional macros that complement the use of certain keys. The rest of this document will outline the other features of the package.

3 List of keys

3.1 Integration limits

3.1.1 Defining the limits

`limits` `limits=` {<mixed-list>}, <keyword>
`limits*` `limits*=` {<mixed-list>}, <keyword>

This key determines the integration limits to be used. In its simplest form, the value of the key takes as its argument a list of elements separated by commas (*comma-separated values* or <csv-list>). This simplification of the input requires a convention: the first and second elements of the list will correspond to the lower and upper limits, respectively. This convention can be reversed using `invert-limits` as seen in section 1.

```
\begin{equation}
  \integral[limits={1, 10}]{f(x)}
\end{equation}
```

$$\int_1^{10} f(x) \, dx \quad (2)$$

The notable advantage of the `limits` key is that you can specify the integration limits for several integrals, and the package automatically typesets the corresponding symbols. To separate pairs of limits, we use a semicolon (*semicolon-separated values* or <ssv-list>).

```
\begin{equation}
  \integral[limits={1, 2; 3, 4}, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\int_1^2 \int_3^4 f(x, y) \, dx \, dy \quad (3)$$

It is also possible to specify predefined limits using keywords (see ??). Finally, the starred variant allows the limits of the integral to be expressed as an interval. The direction of the brackets automatically adjusts itself to the presence of $\pm\infty$.

```
\begin{equation}
  \integral[limits*={1, 10}]{f(x)}
\end{equation}
```

$$\int_{[1,10]} f(x) \, dx \quad (4)$$

^oWe therefore mean by <mixed-list> that `limits` can accept a set of <csv-list> in a more general <ssv-list>.

3.1.2 Modify the style

`limits-mode` `limits-mode=<limits | nolimits>`

`new: v4.1.0` Homonymous key to the one seen in section 1, which allows you to modify the style locally rather than globally.

3.2 Display modes

`mode` `mode=<default | nested | product>`

`new: v3.0.0` *Solely* when the `limits` key is used, it is possible to choose between three distinct display styles. Each of these is fully compatible with the `invert-diff` option.

default Typesets all the symbols, then the integrand, then the variables. As the macro operates by default in this mode, it is not necessary to use the key with this value.

nested Typesets a nested integral where the symbol and integrand alternate before all the variables are written.

product Typesets a product of integrals where the symbol, integrand, and variables alternate.

Of course, nothing prevents you from using the `\integral` macro several times in a row to produce the same result as in the example below. However, for those who would prefer to obtain the result with a single macro, this method is permitted.

Important: for the last two modes, the integral will be *cut off* in a certain way. In order to perform this action correctly, the same method must be used as with `limits`, i.e. by using a semicolon.

MODE **default**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={x, y, z}, mode=default]{xyz}
\end{equation}
```

$$\int_1^2 \int_3^4 \int_5^6 xyz \, dx \, dy \, dz \quad (5)$$

MODE **nested**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={z, y, x}, mode=nested]{x;y;z}
\end{equation}
```

$$\int_1^2 x \int_3^4 y \int_5^6 z \, dz \, dy \, dx \quad (6)$$

MODE **product**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={x, y, z}, mode=product]{x;y;z}
\end{equation}
```

$$\int_1^2 x \, dx \int_3^4 y \, dy \int_5^6 z \, dz \quad (7)$$

To function correctly, it is clear that the keys `limits` and `variables` (see 3.4.1), as well as the integrand, must have the same number of elements.

3.3 Symbol (and more about limits)

The `limits(*)` keys are very useful when typesetting a definite integral. However, this only covers final expressions where the limits of each variable have been specified. For more general cases — indefinite integrals — they are relatively inconvenient. To typeset a double integral over a surface S , one would have to write `limits={,;S,}`. Needless to say, this is rather inconvenient for the user and not optimal for the package. Furthermore, the glyph would not be correct. The set of keys below therefore offers an easy way to modify both the symbol and the bounds.

3.3.1 Selecting a symbol

symbol

`symbol=<control sequence>`

update: v3.0.0

This key accepts a macro designating an integral symbol. Any symbol previously defined by a control sequence is acceptable. If you attempt to use one that is not defined, it will be substituted with `\iint` and a warning message will be issued.

Remark: aside from checking whether the macro exists, no specific checks are performed, and the value of the key is used as-is to typeset the symbol. This implies that the symbol will depend on how it is defined. For example, *amsmath* and *unicode-math* do not define `\iint` in the same way. The result will therefore naturally be different: one combines two `\iint`s with a certain spacing to define `\iint`, while the other defines the actual glyph U+222C.

```
\begin{equation}
  \integral[symbol=\iint, llimit=S, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\iint_S f(x, y) \, dx \, dy \quad (8)$$

3.3.2 Generating many integrals

nint `nint=<integer>`

This key accepts an integer n which allows n integrals to be composed. It is advisable to use this key only if the number of symbols exceeds 4 to favour the glyphs defined by the mathematical font used.

```
\begin{equation}
\int[nint=5, llimit=\Omega, variables={x_1, x_2, x_3, x_4, x_5}]{f(x_1, x_2,
x_3, x_4, x_5)}
\end{equation}
```

$$\iiint\limits_{\Omega} f(x_1, x_2, x_3, x_4, x_5) dx_1 dx_2 dx_3 dx_4 dx_5 \quad (9)$$

3.3.3 Defining boundaries on an ad hoc basis

```
llimit    llimit=<lower limit>
ulimit    ulimit=<upper limit>
```

update: v3.0.0 These two keys allow you to specify the lower and upper limits, respectively. They are only suitable if a single symbol is displayed. If you need to specify both limits, the `limits` key should be used instead.

```
\begin{equation}
\int[llimit={x^2 + y^2 \leq 1}, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\int_{x^2+y^2 \leq 1} f(x, y) dx dy \quad (10)$$

3.3.4 Recurring limits motif

Limits sometimes follow common patterns that can be generalised with keys, thus avoiding overloading the argument of `llimit`.

```
domain    domain= {<*-list>}
domain*    domain*= {<*-list>}
```

update: v3.0.0 These keys accept a list delimited by an asterisk. Each element of the list is then parsed as follows:

- The first token of the item is passed `\mathbb` (preceded by `\uppercase` in case the SHIFT key is too far away for your fingers).
- The remaining tokens — which may be empty — are placed as superscript (or subscript for the starred variant of the key).

```
\begin{equation}
\int[symbol=\iint, domain={\mathbb R*\mathbb R}, variables={x, y}]{xy}
\end{equation}
```

$$\iint_{\mathbb R \times \mathbb R} xy dx dy \quad (11)$$

```
boundary    boundary= {<lower limit>}
```

This key simply places the symbol ∂ before the lower limit.

```
\begin{equation}
  \int_{\partial S} G(\vec{r}) \cdot d\vec{r}
\end{equation}
```

$$\oint_{\partial S} G(\vec{r}) \cdot d\vec{r} \quad (12)$$

3.4 Differentials

3.4.1 Specifying the integration variables

variables `variables={⟨csv-list⟩}, ⟨keyword⟩, none`

update: v3.0.0 This key allows you to define the variables of the integral in the form of a `⟨csv-list⟩`. Like `limits`, the key can accept a keyword as an argument. This behaviour is also explained later (see ??).

Remark: if no variable is specified, i.e. `variables` is not called, the package automatically sets “`dx`” (or “`d\vec{r}`” if `diff-vec` is active). Furthermore, if `none` is passed as a value, no variable will be displayed.

```
\begin{equation}
  \int t^2 dt
\end{equation}
```

$$\int t^2 dt \quad (13)$$

3.4.2 Including the Jacobian

jacobian `jacobian`

This key enables the display of the Jacobian when defined by `\NewVariableKeyword`.

```
\begin{equation}
  \int_0^R \int_0^{2\pi} \int_0^\pi r^2 \sin \theta dr d\theta d\phi
\end{equation}
```

$$\int_0^R r^2 dr \int_0^{2\pi} \sin \theta d\theta \int_0^\pi d\phi \quad (14)$$

3.4.3 Modifying the differential symbol

diff-symb `diff-symb=⟨control sequence⟩`

This key allows you to modify the symbol or style of the differentials.

```
\begin{equation}
  \int \frac{d}{dt} q(t) dt
\end{equation}
```

$$\int e^{+\frac{iS[q(t)]}{\hbar}} \mathcal{D}q(t) \quad (15)$$

3.4.4 Creating a path integral

diff-vec `diff-vec`

This key applies a vector to each integration variable along with a centred dot. It is only compatible with the `default` mode. Using it with the `nested` or `product` options will have no effect.

```
\begin{equation}
W = \int_C \vec{F} \cdot d\vec{s}
\end{equation}
```

$$W = \int_C \vec{F} \cdot d\vec{s} \quad (16)$$

3.4.5 Adding an exponent

diff-order `diff-order=<csv-list>`

This key places an exponent on each differential.

```
\begin{equation}
S[\phi] = \int \mathcal{L}(\phi, \partial_\mu \phi, x^\mu) d^4x
\end{equation}
```

$$S[\phi] = \int \mathcal{L}(\phi, \partial_\mu \phi, x^\mu) d^4x \quad (17)$$

4 Additional macros

In addition to the wide range of keys defined by the package, a few macros also enhance their uses.

4.1 Swapping the position of the differentials

\invertdiff `\invertdiff`

new: v4.1.0

This macro inverts the status of the `invert-diff` option, whether it is initialised to `true` or `false`.

```
\invertdiff
\begin{equation}
\int f(x) dx
\end{equation}
\invertdiff
\begin{equation}
\int f(\omega) d\omega
\end{equation}
```

$$\int dx f(x) \quad (18)$$

$$\int f(\omega) d\omega \quad (19)$$

4.2 Macros contextuelles

The following macros should be used within `\integral`'s main argument. Outside of this context, they will return an error message. They allow, in particular, to include elements defined by the keys directly in the integrand.

`\differentials` `\differentials`

Although the `invert-diff` option exists, it is often desirable to be able to place differentials wherever one wishes. For example, they are frequently seen in the numerator of a fraction when the numerator is 1. To meet this need, the package provides the macro `\differentials`, whose functioning varies slightly from one display mode to another:

- **default**: typesets all the differentials at once. It is compatible with `invert-diff/\invertdiff`.
- **nested**: typesets all the differentials at once. The macro is *not* defined if `invert-diff/\invertdiff` is active.
- **product**: typesets a single differential at a time. It will therefore need to be repeated between each semicolon. It is also compatible with `invert-diff/\invertdiff`.

```
\begin{gather}
\integral{\frac{\differentials}{x}}{\[1cm]}
\langle 0 \mid T\{ \phi(x) \phi(y) \} \mid 0 \rangle = \integral[diff-order=4,
variables=p]{\frac{\differentials}{(2\pi)^4} \frac{e^{-ip \cdot (x-y)}}{p^2 - m
^2 + i \epsilon}}
\end{gather}
```

$$\int \frac{dx}{x} \quad (20)$$

$$\langle 0 \mid T\{\phi(x)\phi(y)\} \mid 0 \rangle = \int \frac{d^4p}{(2\pi)^4} \frac{e^{-ip \cdot (x-y)}}{p^2 - m^2 + i\epsilon} \quad (21)$$

`\jacobian` `\jacobian`

new: v4.2.0

Just like for `\differentials`, this macro allows you to choose the placement of the Jacobian within the integrand. Its functioning also depends on the chosen display mode:

- **default**: typeset the Jacobian in its entirety.
- **nested**: typesets one element of the Jacobian at a time, or in its entirety if `invert-diff/\invertdiff` is active.
- **product**: typesets one element of the Jacobian at a time.

`\variables` `\variables`

new: v4.2.0

This macro allows you to include the variables as defined by the `variables` key directly in the integrand. This prevents you from having to manually rewrite them in the integrand if they are subject to change.

```
\begin{equation}
  \integral[triple=V, variables={x, y, z}]{f(\variables)}
\end{equation}
```

$$\iiint_V f(x, y, z) \, dx \, dy \, dz \quad (22)$$

4.3 Typesetting a primitive

`\antideriv` `\antideriv` [`<list of keys>`] {`<primitive>`}

new: v4.3.0

The package provides only one macro that is not directly related to `\integral`, but which is nevertheless complementary. It allows you to compose a primitive accompanied by delimiters and bounds.

`limits` `limits=` {`<csv-list>`}, `<keyword>`

new: v4.3.0

This key works the same way as the one used in `\integral`, with two exceptions: Cette clé a exactement le même fonctionnement que celle d'`\integral`, à deux exceptions près:

1. The starred variant is not defined.
2. The key expects *only one* pair of bounds. If you use a semicolon, it will be written in the document as it is won't be interpreted.

`delim` `delim=` {`<brackets | bar>`}

new: v4.3.0

This key allows you to locally choose the type of delimiter to use for the primitive.

`big` `big`
`Big` `Big`
`bigg` `bigg`
`Bigg` `Bigg`
`auto` `auto`

new: v4.3.0

These five keys allow you to modify the size of the delimiters. The `auto` key applies `\left` and `\right`.

```
\begin{equation}
\antideriv[limits={0, 1}, delim=bar, Big]{f(x)}
\end{equation}
```

$$f(x) \Big|_0^1 \quad (23)$$

4.4 Looking back at the *limits* key

4.4.1 Defining a keyword

```
\NewLimitsKeyword \NewLimitsKeyword {<keyword>} {<limits>}
\RenewLimitsKeyword
\ProvideLimitsKeyword
\DeclareLimitsKeyword
```

update: v4.0.0

It is common to have to specify the same limits in an integral. To make them easier to write, the `limits(*)` keys accept, in addition to explicit limits, a keyword designating a pair of them predefined by one of these four macros:

- `\NewLimitsKeyword` Creates a new keyword and issues an error if it already exists.
- `\RenewLimitsKeyword` Redefines a keyword and issues an error if it does not already exist.
- `\ProvideLimitsKeyword` Only creates a new keyword if it does not already exist. No message will be issued if this is the case.
- `\DeclareLimitsKeyword` Creates a new keyword regardless, overwriting any previous definition.

When entering a new keyword, you must follow the convention for the order of delimiters. For example, you can write:

```
\usepackage{intexgral}
\NewLimitsKeyword{key1}{A, B}
\intexgralsetup{invert-limits=true}
\NewLimitsKeyword{key2}{B, A}
```

`key1` and `key2` will then produce the same integral. Here is the list of keywords already defined by the package and the limits they contain:

<code>ab</code>	a, b
<code>unit</code>	$0, 1$
<code>real</code>	$-\infty, +\infty$
<code>positive</code>	$0, +\infty$
<code>negative</code>	$-\infty, 0$
<code>circle</code>	$0, 2\pi$
<code>scircle</code>	$0, \pi$
<code>qcircle</code>	$0, \pi/2$
<code>height</code>	$0, H$
<code>radius</code>	$0, R$
<code>length</code>	$0, L$
<code>time</code>	$0, T$

Remark: keywords contain both limits of an integral at the same time. They must therefore be preceded and/or followed by semicolons.

We could therefore modify the expression of an integral as follows:

```
\begin{equation}
  \integral[limits={height;circle;radius}, variables={\rho, \theta, z}]{\rho}
\end{equation}
```

$$\int_0^H \int_0^{2\pi} \int_0^R \rho \, d\rho \, d\theta \, dz \quad (24)$$

It is entirely possible to combine these keywords with the more explicit syntax of `limits`.

```
\begin{equation}
  \integral[limits={height;0,W;length}, variables={x, y, z}]{\kappa(T)\nabla T \cdot \mathbf{n}}
\end{equation}
```

$$\int_0^H \int_0^W \int_0^L \kappa(T) \nabla T \cdot \mathbf{n} \, dx \, dy \, dz \quad (25)$$

4.4.2 Writing symmetric limits

`limits` `limits= {\langle+-limit\rangle}`

new: v4.1.0

When symmetric limits need to be specified, the package can identify them using a syntax specific to the `limits` key. To do this, simply insert the tokens `+-` before the relevant limit.

Remark: the same remark as above applies. Moreover, this syntax *is not* compatible with the keywords seen just before. The goal is to allow the input of occasional symmetric bounds that do not justify the creation of a keyword for a single use.

```
\begin{equation}
  a_n = \frac{1}{\pi} \integral[limits={+-\pi}]{f(x) \cos(x)}
\end{equation}
```

$$a_n = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(x) \cos(x) \, dx \quad (26)$$

4.5 Looking back at the *variables* key

4.5.1 Defining a keyword

```

\NewVariableKeyword \NewVariableKeyword {⟨keyword⟩} {⟨variables⟩} [⟨jacobian⟩]
\RenewVariableKeyword
\ProvideVariableKeyword
\DeclareVariableKeyword

```

update: v4.0.0

Similarly, the same groups of variables often reappear. We can therefore also define keywords containing a set of pre-registered variables. The variants of this macro behave in the same way as for `\NewLimitsKeyword`. The only difference is that here it is possible to define a Jacobian, in the form of a `⟨csv-list⟩` if necessary. Its display is then controlled by the `jacobian` key as explained above. Here is the list of variable keywords already defined by the package, with the Jacobian for some of them:

```

cartesian     $x, y, z$ 
planar        $x, y$ 
polar         $r, \theta (r)$ 
cylindrical   $r, \theta, z (r)$ 
cylindrical*  $\rho, \theta, z (\rho)$ 
spherical     $r, \theta, \phi (r^2, \sin \theta)$ 

```

```

\begin{equation}
\int\limits_V f(r, \theta, \phi) \mathrm{d}r \mathrm{d}\theta \mathrm{d}\phi
\end{equation}

```

$$\iiint_V f(r, \theta, \phi) \mathrm{d}r \mathrm{d}\theta \mathrm{d}\phi \quad (27)$$

4.6 Looking back at the *symbol* key

```

\NewSymbolKeyword \NewSymbolKeyword {\key} {\symbol}
\RenewSymbolKeyword
\ProvideSymbolKeyword
\DeclareSymbolKeyword

```

new: v3.0.0

In order to typeset indefinite integrals, the package essentially offers two keys: `symbol` and `llimit`. Although they are simple to use, it is still laborious to write both of them with their values. This is why *intexgral* provides so-called *shortcut* keys, whose purpose is to combine the selection of the symbol and the limits. Unlike the two previous macros, this involves creating entirely new keys, rather than simply specific values assigned to `symbol`. On their own, these keys modify the integral symbol. The value of these keys corresponds to the lower bound. Here is the list of all *symbol-keys* defined by the package:

<code>single</code>	used symbol = <code>\int</code>
<code>double</code>	used symbol = <code>\iint</code>
<code>triple</code>	used symbol = <code>\iiint</code>
<code>quadruple</code>	used symbol = <code>\iiiiint</code>
<code>contour</code>	used symbol = <code>\oint</code>
<code>surface</code>	used symbol = <code>\oiint</code>
<code>volume</code>	used symbol = <code>\oiiint</code>

Important: allowing users to create their own keys for the `\integral` macro poses a risk that will be explained in more detail in the next section. For now, just remember that all these macros¹ will issue a warning message if you attempt to create a new *symbol-key* with the same name as a group of defined limits.

```

\begin{gather}
\integral[double=S, variables=planar]{f(x, y)}{\[1cm]}
\integral[contour=\mathcal{C}, diff-vec]{f(\vec{r})}
\end{gather}

```

$$\iint_S f(x, y) \, dx \, dy \quad (28)$$

$$\oint_{\mathcal{C}} f(\vec{r}) \cdot d\vec{r} \quad (29)$$

5 Special syntax

5.1 Presentation of the syntax

`\integral` `\integral` [`\limits:variables(+j):mode`] {`\integrand`}

new: v3.0.0

With regard to definite integrals, the user will be required to use a maximum of four keys to compose them: `limits`, `variables`, `mode` and `jacobian`. However, they can be criticised for the same reason as before; writing these four keys, which can take quite long arguments, runs counter to the main objective of this package. Thus, the user can designate as an optional argument of `\integral` a syntax called *special* that is unique to `intexgral`. The logic is as follows:

- You specify a *valid* argument for the `limits` key.
- You specify a *valid* argument for the `variables` key.
- You specify a *valid* argument for the `mode` key.
- And you separate everything with a colon.

Let's look at an example to better understand.

```
\begin{equation}
  \integral[1, 2; 4, 5:y, x:nested]{x; y}
\end{equation}
```

$$\int_1^2 x \int_4^5 y \, dy \, dx \quad (30)$$

We mean by *valid argument* that all forms of values accepted by the four keys can also be likewise adopted by the special syntax. This hence includes keywords.

```
\begin{equation}
  \integral[radius;circle;scircle:spherical:product]{r;\theta;\phi}
\end{equation}
```

$$\int_0^R r \, dr \int_0^{2\pi} \theta \, d\theta \int_0^\pi \phi \, d\phi \quad (31)$$

To include the Jacobian, simply write `+j` after the argument of `variables`. The mode can also be indicated using only an initial.

```
\begin{equation}
  \integral[radius;circle;scircle:spherical+j:p]{;};
\end{equation}
```

$$\int_0^R r^2 \, dr \int_0^{2\pi} \sin \theta \, d\theta \int_0^\pi d\phi \quad (32)$$

It is important to reiterate that in the classic configuration of the optional argument, it is of course not mandatory to specify the four keys mentioned. The same applies to this syntax. Since the key `mode` is not necessary for `default`, this part can be omitted.

```
\begin{equation}
\integral[1, 2; 3, 4:x, y]{xy}
\end{equation}
```

$$\int_1^2 \int_3^4 xy \, dx \, dy \quad (33)$$

You can also take advantage of the automatic placement of “ dx ” with this syntax, ignoring `variables`.

```
\begin{equation}
\integral[1, 10]{x}
\end{equation}
```

$$\int_1^{10} x \, dx \quad (34)$$

One notable special case of this syntax is that if you want to modify the variable without specifying the bounds, you can do so more quickly as follows:

```
\begin{equation}
\integral[:z]{f(z)}
\end{equation}
```

$$\int f(z) \, dz \quad (35)$$

Even if it is not recommended.

5.2 Functioning of the syntax

From the user’s perspective, it is relatively simple to choose which syntax to adopt. However, the package must be able to distinguish between the two. Without going too deeply into the details² of the implementation, it is worth noting that the package attempts to extract the first key from the optional argument³ and checks if it exists. This is why, if you create a key with `\NewSymbolKeyword` whose name may probably serve as integration bounds, `intexgral` may falsely evaluate the nature of the optional argument. In this kind of situation, the interpretation of the keys will be incorrect and the result will be wrong.

²The more courageous among you are still invited to read the source code.

³Or at least, the group of tokens it thinks correspond to a key.

6 Additional parameters

`\IntegralSetup` `\IntegralSetup {⟨list of parameters⟩}`

new: v3.0.0
update: v4.0.0

This macro allows, under the syntax `⟨key=value⟩`, to modify parameters related to certain keys. All assignments made are local and the macro can therefore be placed in a group if needed. Here is an example of the macro with the default assignments of the package:

```
\IntegralSetup{
  diff-symb      = \l_@@_default_differential_symbol_tl, % see at the beginning of the
                  implementation
  defaultvar     = {x},
  defaultvar*    = {r},
  varsep        = \thinmuskip,
  diffsep       = \thinmuskip,
  innersymbsep   = {-5mu},
  postsymbsep    = {-1mu},
  jacobiansep    = {0mu},
  vectorstyle    = \vec,
  domainstyle    = \mathbb,
  delim         = brackets
}
```

`diff-symb` `diff-symb=⟨control sequence⟩`

new: v4.0.0

This key controls the symbol used for differentials. If the option `italic` is used, `\mathnormal` is applied (see at the beginning of the implementation). Unlike the homonymous key of `\integral`, this one acts at a more global level.

`defaultvar` `defaultvar={⟨csv-list⟩}, ⟨keyword⟩, none`
`defaultvar*`

new: v3.0.0

This key modifies the variable automatically inserted when `variables` is not used. The argument may correspond to a keyword defined by `\NewVariableKeyword`. The value may also be `none` to disable the automatic insertion of the variable. The starred variant of the key controls the integration parameter to be typeset with `diff-vec`.

`varsep` `varsep=⟨muexpr⟩`

new: v4.0.0

This key modifies the spacing between the integrand (or the Jacobian if it is displayed) and the variables.

`diffsep` `diffsep=⟨muexpr⟩`

new: v4.0.0

This key modifies the spacing between the differentials themselves.

`innersymbsep` `innersymbsep=⟨muexpr⟩`

new: v4.0.0

When the `default` mode is in effect, that is to say when the `limits` key is used, the package inserts a gap between the symbols to slightly bring them together. This key therefore allows you to adjust this spacing.

`postsymbsep` `postsymbsep=⟨muexpr⟩`

new: v4.0.0

This key controls the spacing between the symbol and the element that follows (integrand or differential depending on `invert-limits`).

`jacobiansep` `jacobiansep=⟨muexpr⟩`

new: v4.2.0

This key controls the spacing between the integrand and the Jacobian.

vectorstyle `vectorstyle=<control sequence>`

new: v3.0.0 This key modifies the macro to be applied to variables when `diff-vec` is in action. It is therefore possible to alter the style of the vector: bold, underlined, or even another style from a particular package, `esvect` for example.

domainstyle `domainstyle=<control sequence>`

new: v3.0.0 Similarly, it is possible to change the macro to be applied for the `domain(*)` keys.

delim `delim=<brackets | bar>`

new: v4.3.0 This key allows you to choose the type of delimiter to use for the primitives.

Changelog

4.3.1 (21-09-2026)

Fixed

- ▶ Display issue between `nint` and `limits-mode` ([issue #9](#)).

4.3.0 (18-09-2026)

Added

- ▶ Macro `\antideriv`.
- ▶ Keys `delim`, `limits`, `big`, `Big`, `bigg`, `Bigg` and `auto` for `\antideriv`.

Fixed

- ▶ Issue with `defaultvar` ([issue #20](#)).

Changed

- ▶ `defaultvar` can now accept `none` as a value.

Deprecated

- ▶ Key `novar`.

4.2.0 (10-09-2026)

Added

- ▶ `jacobiansep` key.
- ▶ Macros `\jacobian` and `\variables`.

4.1.1 (01-08-2026)

Fixed

- ▶ Issue with `\differentials` used in `product` mode ([issue #14](#)).
- ▶ Issue with `\RenewSymbolKeyword` and `\DeclareSymbolKeyword` ([issue #15](#)).
- ▶ Issue with `\RenewVariableKeyword` and `\DeclareVariableKeyword` ([issue #17](#)).

4.1.0 (27-07-2026)

Added

- ▶ `limits-mode` key for `\integral`.
- ▶ `\invertdiff` macro.
- ▶ The `+-` syntax for limits.

Changed

- ▶ Source code optimisation.

4.0.0 (06-06-2026)

Added

- ▶ Detailed implementation.
- ▶ Keys `varsep`, `diffsep`, `postsymbsep` and `diff-order`.

- ▶ Variable keyword `cylindrical*`.

Changed

- ▶ The `derivative` is no longer a dependency. The differentials are now managed internally.
- ▶ Key `symbolskip`, renamed in `innersymbsep`.
- ▶ Key `hide-diff`, renamed in `novar`.

Removed

- ▶ Keys `diff-star` and `diff-options`.

3.0.1 (02-01-2026)

Fixed

- ▶ Bug with jacobian in special syntax ([issue #3](#)).
- ▶ French and English documentations ([issue #4](#), [issue #6](#) and [issue #7](#)).

Changed

- ▶ Limits keywords `positive` and `real` now contain a `+` sign ([issue #5](#)).

3.0.0 (24-12-2025)

Added

- ▶ Special syntax.
- ▶ Keys `domain*` and `mode`.
- ▶ Macros `\IntegralSetup` and `\NewSymbolKeyword`.

Removed

- ▶ Macros `\defaultdiff`, `\defaultdiff`, `\defaultvdiff` and `\vdiffstyle` in favour of `\IntegralSetup`.
- ▶ Keys controlling both symbol and limit, now managed at a higher level by `\NewSymbolKeyword`.
- ▶ `int-split` key, in favour of `mode`.
- ▶ `\NewIntegralSymbol` macro.

Changed

- ▶ The names of some keys (`variable` to `variables`, `lower-lim` and `upper-lim` to `llimit` and `ulimit`, `int-symb` to `symbol`, `invert-differentials` and `hide-differentials` to `invert-diff` and `hide-diff`).
- ▶ `jacobian` and `diff-star` no longer require a boolean value. Simply writing them will now enable their features.
- ▶ `hide-diff` assigned to a local option rather than a package one.
- ▶ `limits-mode` assigned to a package option rather than a macro key

2.0.1 (13-09-2025)

Fixed

- ▶ Compatibility issue between `unicode-math` and `amssymb` depending on the loading order ([problem #2](#)).

2.0.0 (09-09-2025)

Added

- ▶ Macros `\intexgralsetup`, `\defaultdiff`, `\defaultvdiff` and `\vdiffstyle`.

Changed

- ▶ Warning messages related to non-existing symbols. They are now only triggered when the integral is typeset.

Removed

- ▶ `diff-vec-style` key in favour of `\vdiffstyle`.
- ▶ Warning message about misuse of semi-colon in conjunction with the `int-split` key.

Fixed

- ▶ Bug where the integrand was not reset when `\integral` was used successively in the same $\text{\TeX}$ group ([details here](#)).
- ▶ `expl3` log leftovers that shouldn't have been there.

1.1.0 (29-07-2025)

Added

- ▶ Starred variant for the keys controlling both symbol and limit (keys `single`, `contour` etc).

1.0.0 (26-07-2025) — Initial version.

PART II

IMPLEMENTATION

1 Initialisation

```
1 <*package>
2 <@@=intexgral>
```

1.1 Package declaration

```
3 \NeedsTeXFormat{LaTeX2e}
4 \RequirePackage{expl3}[2025-05-14]
5
6 \def\intexgral@module{intexgral}
7 \def\intexgral@version{4.3.1}
8 \def\intexgral@date{2026-09-21}
9 \def\intexgral@description{A LaTeX package for typesetting integrals}
10
11 \ProvidesExplPackage
12   \intexgral@module
13   \intexgral@date
14   \intexgral@version
15   \intexgral@description
```

1.2 Preliminary verifications

We check that the version of `expl3` is recent enough before continuing. The package requires the macro `\tl_if_regex_match:nnTF`, available from December 2024.

```
16 \msg_new:nnn { intexgral } { expl3-too-old }
17 {
18   Your~expl3~installation~is~too~old,~
19   "intexgral"~requires~expl3~dated~2024-12-08~or~later.\iow_newline:
20   Package~loading~has~been~aborted.\iow_newline:
21   \msg_see_documentation_text:n { intexgral }
22 }
23
24 \@ifpackagelater{expl3}{2024-12-08}{}
25 { \msg_critical:nn { intexgral } { expl3-too-old } }
```

Since `intexgral` uses `\mathbb` for the keys `domain(*)`, we must ensure that the macro exists. The verification is placed in a *hook* executed at the beginning of the document in case `amsfonts` or any package defining `\mathbb` is loaded after `intexgral`. This serves particularly to avoid conflicts between packages (`unicode-math` and `amssymb` for example).

```
26 \hook_gput_code:nnn { begindocument/before } { . }
27 { \cs_if_exist:NF \mathbb { \RequirePackage{amsfonts} } }
```

2 Package options

2.1 Variable definition

`\l_@@_invert_limits_bool`

Boolean inverting the order of the limits.

```
28 \bool_new:N \l__intexgral_invert_limits_bool
```

`\l_@@_deactivate_variables_bool`

Determines whether the differentials should be displayed or not.

```
29 \bool_new:N \l__intexgral_deactivate_variables_bool
```

`\l_@@_invert_differential_bool`

Determines the order of display for the differentials.

```
30 \bool_new:N \l__intexgral_invert_differential_bool
```

`\l_@@_default_differential_symbol_tl`

Contains the default symbol for differentials.

```
31 \tl_new:N \l__intexgral_default_differential_symbol_tl
```

`\l_@@_limits_style_tl`

Contains the $\TeX$ primitive to use for the limits style.

```
32 \tl_new:N \l__intexgral_limits_style_tl
```

2.2 Package option declaration

We can now define the package options before loading them.

```
33 \keys_define:nn { intexgral }
34   {
35     invert-limits .bool_set:N = \l__intexgral_invert_limits_bool,
36     invert-limits .usage:n    = { preamble },
37     invert-limits .initial:n  = { false },
38
39     invert-diff .bool_set:N = \l__intexgral_invert_differential_bool,
40     invert-diff .usage:n    = { preamble },
41     invert-diff .initial:n  = { false },
42
43     limits-mode .choice:,
44     limits-mode / limits .code:n =
45       { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_limits:D },
46     limits-mode / nolimits .code:n =
47       { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_nolimits:D },
48
49     limits-mode .usage:n    = { preamble },
50     limits-mode .initial:n  = { nolimits },
51
52     italic .choice:,
53     italic / true .code:n =
54       {
55         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
```

```

56         { \mathnormal{d} }
57     },
58     italic / false .code:n =
59     {
60         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
61         { \mathrm{d} }
62     },
63     italic .usage:n = { preamble },
64     italic .initial:n = { false },
65
66     upright .choice:,
67     upright / true .code:n =
68     {
69         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
70         { \mathrm{d} }
71     },
72     upright / false .code:n =
73     {
74         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
75         { \mathnormal{d} }
76     },
77     upright .usage:n = { preamble },
78     upright .initial:n = { true }
79 }
80
81 \ProcessKeyOptions[intexgral]

```

3 Messages

The next part of the implementation defines all the warning and error messages.

`\g_@@_integral_number_int`

We start by creating a global counter to number the integrals in the document.

```
82 \int_gzero_new:N \g__intexgral_integral_number_int
```

`\@@_warning_msg_header: ★`

We can now define a macro that acts as a header to warning messages. It uses the counter defined just before.

```

83 \cs_new:Nn \__intexgral_warning_msg_header: {
84     (
85         integral~no.~
86         \int_use:N\g__intexgral_integral_number_int\c_space_tl
87         \msg_line_context:
88     )
89     \iow_newline:
90 }

```

All the messages are then defined.

```

91 \msg_new:nnnn { intexgral } { symb-key-alr-def }
92 { Symbol~key~"\tl_trim_spaces:n{#1}"~is~already~defined. }
93 { Use~\token_to_str:N \RenewSymbolKeyword\ instead. }
94
95 \msg_new:nnnn { intexgral } { symb-key-not-def }
96 { Symbol~key~"\tl_trim_spaces:n{#1}"~is~not~defined. }
97 { Use~\token_to_str:N \NewSymbolKeyword\ instead. }

```

```

98
99 \msg_new:nnnn { intexgral } { diff-group-alr-def }
100   { Differential~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
101   { Use~\token_to_str:N \RenewVarKeyword\ instead. }
102
103 \msg_new:nnnn { intexgral } { diff-group-not-def }
104   { Differential~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
105   { Use~\token_to_str:N \NewVarKeyword\ instead. }
106
107 \msg_new:nnnn { intexgral } { lim-group-alr-def }
108   { Limits~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
109   { Use~\token_to_str:N \RenewLimitsKeyword\ instead. }
110
111 \msg_new:nnnn { intexgral } { lim-group-not-def }
112   { Limits~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
113   { Use~\NewLimitsKeyword\ instead. }
114
115 \msg_new:nnn { intexgral } { key-exists-for-lim }
116   {
117     Symbol~key~already~defined~with~
118     \token_to_str:N \NewLimitsKeyword\ \__intexgral_warning_msg_header:
119     Key~"#1"~already~exists~for~predefined~limits.~
120     This~can~disrupt~the~functioning~of~the~special~syntax.
121   }
122
123 \msg_new:nnn { intexgral } { unknown-symb }
124   {
125     Unknown~integral~symbol~\__intexgral_warning_msg_header:
126     The~symbol~\tl_trim_spaces:n{#1}~has~been~replaced~with~
127     \token_to_str:N\int.
128   }
129
130 \msg_new:nnn { intexgral } { no-jacobian }
131   {
132     Jacobian~unavailable~\__intexgral_warning_msg_header:
133     A~Jacobian~was~requested~for~the~"#1"~keyword,
134     ~but~none~was~declared~for~the~latter.
135   }
136
137 \msg_new:nnn { intexgral } { use-glyph-instead }
138   {
139     Consider~using~the~dedicated~glyph~\__intexgral_warning_msg_header:
140     The~key~"nint"~was~used~with~fewer~than~
141     5~regular~integral~signs.
142   }
143
144 \msg_new:nnn { intexgral } { differentials-outside }
145   { \cs_to_str:N\differentials~cannot~appear~outside~of~an~integral. }
146
147 \msg_new:nnn { intexgral } { jacobian-outside }
148   { \cs_to_str:N\jacobian~cannot~appear~outside~of~an~integral. }
149
150 \msg_new:nnn { intexgral } { variables-outside }
151   { \cs_to_str:N\variables~cannot~appear~outside~of~an~integral. }
152
153 \msg_new:nnn { intexgral } { novar-deprecated }
154   {
155     The~key~"novar"~is~deprecated~\__intexgral_warning_msg_header:

```

```

156         Use~"defaultvar=none"~instead.
157     }

```

4 Useful variants

```

\str_if_eq:neTF
\str_if_eq:neF
\keys_if_exist:nVTF
\seq_use:Ne
\str_if_eq_p:en
\prop_gput:Nne
\seq_gset_split:cnn
\seq_set_split:Nnf

```

```

158 \cs_generate_variant:Nn \str_if_eq:nnTF { neTF }
159 \cs_generate_variant:Nn \str_if_eq:nnF { neF }
160 \cs_generate_variant:Nn \keys_if_exist:nnTF { nVTF }
161 \cs_generate_variant:Nn \seq_use:Nn { Ne }
162 \cs_generate_variant:Nn \str_if_eq_p:nn { en }
163 \cs_generate_variant:Nn \prop_gput:Nnn { Nne }
164 \cs_generate_variant:Nn \seq_gset_split:Nnn { cnn }
165 \cs_generate_variant:Nn \seq_set_split:Nnn { Nnf }

```

5 Internal variables

5.1 Tokens

```

\l_@@_integrand_tl

```

Token containing the integrand. It is used as-is with the `default` mode, whereas for the other modes, it is subsequently separated into a sequence.

```

166 \tl_new:N \l__intexgral_integrand_tl

```

```

\l_@@_integral_symbol_tl

```

Token containing the integral symbol. When not modified by the `symbol` key, it should still be initialised to `\int`.

```

167 \tl_new:N \l__intexgral_integral_symbol_tl
168 \tl_set_eq:NN \l__intexgral_integral_symbol_tl \int

```

```

\l_@@_lower_limit_tl
\l_@@_upper_limit_tl

```

Tokens containing the limits.

```

169 \tl_new:N \l__intexgral_lower_limit_tl
170 \tl_new:N \l__intexgral_upper_limit_tl

```

```

\l_@@_differential_symbol_tl

```

Token containing the symbol for differentials, which can be modified with the `diff-symb` key. It is initialised to the `\l_@@_default_differential_symbol_tl` token.

```

171 \tl_new:N \l__intexgral_differential_symbol_tl

```

`\l_@@_vectorial_differential_style_tl`
`\l_@@_domain_style_tl`

Tokens containing the macros to apply to the `diff-vec` and `domain(*)` keys.

```
172 \tl_new:N \l__integragl_vectorial_differential_style_tl
173 \tl_new:N \l__integragl_domain_style_tl
```

`\l_@@_domain_char_tl`
`\l_@@_domain_dimension_tl`

Token used for storing respectively the character and the dimension of a domain.

```
174 \tl_new:N \l__integragl_domain_char_tl
175 \tl_new:N \l__integragl_domain_dimension_tl
```

`\c_@@_left_bracket_tl`
`\c_@@_right_bracket_tl`

Constant tokens for the `limits*` key.

```
176 \tl_const:Nn \c__integragl_left_bracket_tl { [ }
177 \tl_const:Nn \c__integragl_right_bracket_tl { ] }
```

`\l_@@_key_name_tl`

Token containing the potential name of a key, used to differentiate the syntax `<key=value>` from the special syntax.

```
178 \tl_new:N \l__integragl_key_name_tl
```

`\l_@@_primitive_tl`

Token containing the primitive.

```
179 \tl_new:N \l__integragl_primitive_tl
```

`\l_@@_primitive_left_delim_tl`
`\l_@@_primitive_right_delim_tl`

Tokens containing the delimiters of the primitive.

```
180 \tl_new:N \l__integragl_primitive_left_delim_tl
181 \tl_new:N \l__integragl_primitive_right_delim_tl
```

`\l_@@_primitive_left_delim_size_tl`
`\l_@@_primitive_right_delim_size_tl`

Tokens containing the size of the delimiters of the primitive.

```
182 \tl_new:N \l__integragl_primitive_left_delim_size_tl
183 \tl_new:N \l__integragl_primitive_right_delim_size_tl
```

5.2 Booleans

`\l_@@_manual_differential_bool`

Boolean used to determine if the differentials/jacobian are inserted through `\differentials/``\jacobian`.

```
184 \bool_new:N \l__integragl_manual_differential_bool
185 \bool_new:N \l__integragl_manual_jacobian_bool
```

`\l_@@_custom_variables_bool`

Boolean used to determine if the differentials are inserted through the `variables` key.

```
186 \bool_new:N \l__integragl_custom_variables_bool
```

`\l_@@_separate_integral_bool`

Boolean used to determine if the integral is *separated* into several parts, that is to say, if the modes `nested` or `product` are in effect.

187 `\bool_new:N \l__intexgral_separate_integral_bool`

`\l_@@_vectorial_differential_bool`

Boolean used to determine if vectors should be applied to the differentials.

188 `\bool_new:N \l__intexgral_vectorial_differential_bool`

`\l_@@_jacobian_bool`

Boolean used to determine if the Jacobian should be displayed.

189 `\bool_new:N \l__intexgral_jacobian_bool`

5.3 Sequences and clists

`\l_@@_domain_seq`

Sequence containing the integration domains and their dimensions (the main sequences will be explained later).

190 `\seq_new:N \l__intexgral_domain_seq`

`\l__intexgral_default_variables_seq`

`\l__intexgral_default_vectorial_variables_seq`

Sequences containing the default variables, used when the `variables` key is not specified.

191 `\seq_new:N \l__intexgral_default_variables_seq`

192 `\seq_new:N \l__intexgral_default_vectorial_variables_seq`

`\l_@@_differential_order_clist`

List containing the powers of the differentials (clist and not sequence since `.seq_set:N` does not exist for keys).

193 `\clist_new:N \l__intexgral_differential_order_clist`

5.4 Property lists

`\g_@@_limits_keyword_prop`

`\g_@@_differential_group_keyword_prop`

A property list is created to store limits and variable keywords.

194 `\prop_new:N \g__intexgral_limits_keyword_prop`

195 `\prop_new:N \g__intexgral_differential_group_keyword_prop`

5.5 Muskip

```
\l_@@_symbol_inner_sep_muskip
\l_@@_symbol_post_sep_muskip
\l_@@_differential_sep_muskip
\l_@@_variable_sep_muskip
\l_@@_jacobian_sep_muskip
```

Internal muskips used to store the spacings for respectively:

1. The spacing between the symbols when several are generated by `limits(*)`.
2. The spacing that follows the symbol.
3. The spacing between the differentials when the key `variables` is used.
4. The spacing between the differentials and the integrand.
5. The spacing between the integrand and the Jacobian.

```
196 \muskip_new:N \l__intexgral_symbol_inner_sep_muskip
197 \muskip_new:N \l__intexgral_symbol_post_sep_muskip
198 \muskip_new:N \l__intexgral_variable_sep_muskip
199 \muskip_new:N \l__intexgral_differential_sep_muskip
200 \muskip_new:N \l__intexgral_jacobian_sep_muskip
```

5.6 String

```
\l_@@_display_mode_str
```

Internal variable used to store the integral's display mode. The reason for using a *string* stems from the lack of a `\tl_case:nnTF` macro.

```
201 \str_new:N \l__intexgral_display_mode_str
```

6 Auxiliary macros and sequences

Here we define the two $\TeX$ primitives `\mkern` and `\mathchoice` with an *expl3*-like syntax.

```
\@@_mkern:N \@@_mkern:N <muskip>
```

```
202 \cs_new_protected:Npn \__intexgral_mkern:N #1
203 { \tex_mkern:D #1 \scan_stop: }
```

```
\@@_mathchoice:nnnn \@@_mathchoice:nnnn {\display style} {\inline style} {\script style}
{\scriptscript style}
```

```
204 \cs_new_eq:NN \__intexgral_mathchoice:nnnn \mathchoice
```

`\@@_invert_limits:w` `\@@_invert_limits:w <lower limit>,<upper limit>\q_stop`

Macro inverting the limits for the `invert-limits` option. The action is fully expandable and is more efficient than a `\seq_inverse:N` in the context of this package, knowing that a maximum of two elements are present.

```
205 \cs_new:Npn \__intexgral_invert_limits:w #1,#2\q_stop{#2,#1}
```

`\@@_general_integral_symbol:`

We create a macro that contains the general expression for an integral: symbol, limits style, lower limit and upper limit. We just have to assign the tokens to compose a complete symbol, and possibly repeat the macro for integrals defined on several variables.

```
206 \cs_new:Npn \__intexgral_general_integral_symbol:
207   {
208     \tex_mathop:D
209     {
210       \l__intexgral_integral_symbol_tl
211       \l__intexgral_limits_style_tl
212       \c_math_subscript_token
213       { \l__intexgral_lower_limit_tl }
214       \c_math_superscript_token
215       { \l__intexgral_upper_limit_tl }
216     }
217   }
```

6.1 Main sequences

Given the large number of existing keys and possible combinations, the tokens must be carefully organized and stored in order to assemble the final integral. The approach taken was to create sequences dedicated to each element of an integral:

`\l__integral_symbol_seq`

One for the symbol (or the *symbols*, if the key `limits*` is in use).

```
218 \seq_new:N \l__intexgral_integral_symbol_seq
```

`\l__integrand_seq`

One for the integrand (or *the* integrands, if split into several parts by `nested` or `product`).

```
219 \seq_new:N \l__intexgral_integrand_seq
```

`\l__jacobian_seq`

One for the Jacobian.

```
220 \seq_new:N \l__intexgral_jacobian_seq
```

`\l__differential_seq`

And finally, one for the differentials.

```
221 \seq_new:N \l__intexgral_differential_seq
```

This way, we can construct the final integral by varying how we extract the different elements from these sequences. Here is a more concrete example:

For an indefinite integral.

$$\boxed{\int\!\!\int_S} \boxed{f(x,y)} \boxed{dx\,dy}$$

For a definite integral.

$$\boxed{\int_a^b} \boxed{\int_c^d} \boxed{f(x)} \boxed{g(y)} \boxed{dx} \boxed{dy}$$

6.2 Supplementary sequences

A few additional sequences will be required during the intermediate stages, which will involve preparing the four main sequences listed above.

`\l_@@_variables_seq`

Firstly, a sequence containing the variables.

```
222 \seq_new:N \l__intexgral_variables_seq
```

`\l_@@_limits_seq`

Then, a sequence containing the pairs of integration limits, that is to say the elements of the `limits` key separated in two by the semicolon.

```
223 \seq_new:N \l__intexgral_limits_seq
```

`\l_@@_upper_limits_seq`

`\l_@@_lower_limits_seq`

Two sequences that contain the upper and lower limits respectively. Thus, if the key `limits` contains the elements `a;b, c;d` and `e;f`, then the sequence of lower limits will contain the elements `a, c` and `e`, while the sequence of upper limits will contain the elements `b, d` and `f`.

```
224 \seq_new:N \l__intexgral_upper_limits_seq
```

```
225 \seq_new:N \l__intexgral_lower_limits_seq
```

7 Integration limits

`\@@_has_pm_p:n`

We define a conditional macro that will handle the `+-<limit>` syntax. It simply checks that the first and second tokens of the `limits` key argument correspond respectively to `+` and `-`.

```
226 \prg_new_conditional:Npnn \__intexgral_has_pm:n #1 { p }
227 {
228   \bool_lazy_and:nnTF
229     { \exp_args:Ne \str_if_eq_p:nn { \str_item:nn { #1 } { 1 } } { + } }
230     { \exp_args:Ne \str_if_eq_p:nn { \str_item:nn { #1 } { 2 } } { - } }
231     { \prg_return_true: }
232     { \prg_return_false: }
233 }
```

`\@@_trim_pm:w`

Then, a delimited macro is created to remove the +-

```
234 \cs_new:Npn \__intexgral_trim_pm:w +-#1\q_stop{#1}
```

`\@@_parse_limits:n` ★

`\@@_parse_limits:n {<lower limit>} {< upper limit>}, <keyword>`

This macro executes one of the following actions depending on the boolean value that is true:

- If the argument is a keyword defined for limits, the macro expands to its content, inverting it if necessary (depending on the status of `\l_@@_invert_limits_bool`).
- If the argument starts with the tokens `+-`, removes them and adds the signs manually.
- If none of these two conditions are met, simply leaves the argument in the input stream.

```
235 \cs_new:Npn \__intexgral_parse_limits:n #1
236 {
237   \bool_case:nF
238   {
239     { \prop_if_in_p:Nn \g__intexgral_limits_keyword_prop { #1 } }
240     {
241       \bool_if:NTF \l__intexgral_invert_limits_bool
242       {
243         \exp_last_unbraced:Ne \__intexgral_invert_limits:w
244         { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
245         \q_stop
246       }
247       { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
248     }
249     { \__intexgral_has_pm_p:n { #1 } }
250     {
251       -\__intexgral_trim_pm:w #1\q_stop,
252       +\__intexgral_trim_pm:w #1\q_stop
253     }
254   }
255   { #1 }
256 }
```

`\@@_set_limits_regular:`

Once the argument of the key `limits` has been converted into a sequence (action performed within `.code:n` in the key declaration), each pair of limits is assigned to a temporary sequence. This sequence is created so that if a keyword is present, it is replaced by its corresponding limits according to the previous macro. The action is executed within a `\romannumeral`-type expansion to avoid expanding limits containing fragile macros (such as `\mathbb`). The lower and upper limits can then be extracted from this sequence and assigned to their respective sequences.

```
257 \cs_new_protected:Nn \__intexgral_set_limits_regular:
258 {
259   \seq_map_inline:Nn \l__intexgral_limits_seq
260   {
261     \seq_set_split:Nnf \l_tmpa_seq { , }
```

```

262         { \__intexgral_parse_limits:n { ##1 } }
263     \seq_put_right:Ne \l__intexgral_lower_limits_seq
264         { \seq_item:Nn \l_tmpa_seq { 1 } }
265     \seq_put_right:Ne \l__intexgral_upper_limits_seq
266         { \seq_item:Nn \l_tmpa_seq { 2 } }
267 }
268 }

```

\@@_set_limits_starred:

The following macro is similar to the previous one, but it handles limits in the form of intervals when the key `limits*` is used.

```

269 \cs_new_protected:Npn \__intexgral_set_limits_starred:
270 {
271     \seq_map_indexed_inline:Nn \l__intexgral_limits_seq
272     {
273         \seq_set_split:Nnf \l_tmpa_seq { , }
274         { \__intexgral_parse_limits:n { ##2 } }
275         \bool_if:NTF \l__intexgral_invert_limits_bool
276         {
277             \tl_set:Ne \l__intexgral_lower_limit_tl
278             {
279                 \seq_item:Nn \l_tmpa_seq { 2 }
280                 \tex_mathpunct:D,
281                 \seq_item:Nn \l_tmpa_seq { 1 }
282             }
283         }
284         {
285             \tl_set:Ne \l__intexgral_lower_limit_tl
286             {
287                 \seq_item:Nn \l_tmpa_seq { 1 }
288                 \tex_mathpunct:D,
289                 \seq_item:Nn \l_tmpa_seq { 2 }
290             }
291         }
292         \bool_if:NTF \l__intexgral_invert_limits_bool
293         { \seq_put_right:Ne \l__intexgral_upper_limits_seq }
294         { \seq_put_right:Ne \l__intexgral_lower_limits_seq }
295         {
296             \str_case_e:nnF { \seq_item:Nn \l_tmpa_seq { 1 } }
297             { { -\infty } { \tex_left:D \c__intexgral_right_bracket_tl } }
298             { \tex_left:D \c__intexgral_left_bracket_tl }
299             \tl_use:N \l__intexgral_lower_limit_tl
300             \str_case_e:nnF { \seq_item:Nn \l_tmpa_seq { 2 } }
301             { { +\infty } { \tex_right:D \c__intexgral_left_bracket_tl } }
302             { \tex_right:D \c__intexgral_right_bracket_tl }
303         }
304     }
305 }

```

\@@_set_limits:nn \@@_set_limits:nn {<lower limit>} {<upper limit>}

According to the `invert-limits` option, this macro assigns the lower and upper limits to their respective tokens. If it is active, the assignment becomes somewhat counterintuitive (the lower limit is assigned to `\l__@@_upper_limit_tl` and vice versa), but

it will still produce the expected result. The expansion of the limits is prevented to avoid issues with fragile macros as seen previously.

```

306 \cs_new_protected:Npn \__intexgral_set_limits:nn #1#2
307 {
308     \bool_if:NTF \l__intexgral_invert_limits_bool
309     {
310         \tl_set:Nx \l__intexgral_lower_limit_tl { \exp_not:n { #2 } }
311         \tl_set:Nx \l__intexgral_upper_limit_tl { \exp_not:n { #1 } }
312     }
313     {
314         \tl_set:Nx \l__intexgral_lower_limit_tl { \exp_not:n { #1 } }
315         \tl_set:Nx \l__intexgral_upper_limit_tl { \exp_not:n { #2 } }
316     }
317 }

```

\@@_generate_integral_sequence:

Once the sequences of lower and upper limits are filled, we can use the structure of the sequences to our advantage to automatically generate as many symbols as necessary. We proceed with an incremented loop in which we perform two actions:

1. With the help of `\@@_set_limits:nn`, we assign the integration limits to the `\l_@@_lower_limit_tl` and `\l_@@_upper_limit_tl` tokens from the i^{th} element of the `\l_@@_lower_limits_seq` and `\l_@@_upper_limits_seq` sequences respectively.
2. We place to the right (i.e. in order) of the `\l_@@_integral_symbol_seq` sequence the `\@@_general_integral_symbol:` macro, which contains the general form of an integral. Note that we fully expand its content in order to retrieve the immediate assignment of tokens.

```

318 \cs_new_protected:Nn \__intexgral_generate_integral_sequence:
319 {
320     \int_set:Nn \l_tmpa_int
321     {
322         \seq_if_empty:NTF \l__intexgral_lower_limits_seq
323         { \seq_count:N \l__intexgral_upper_limits_seq }
324         { \seq_count:N \l__intexgral_lower_limits_seq }
325     }
326     \int_step_inline:nn { \l_tmpa_int }
327     {
328         \__intexgral_set_limits:nn
329         { \seq_item:Nn \l__intexgral_lower_limits_seq { ##1 } }
330         { \seq_item:Nn \l__intexgral_upper_limits_seq { ##1 } }
331         \seq_put_right:Nx \l__intexgral_integral_symbol_seq
332         { \__intexgral_general_integral_symbol: }
333     }
334 }

```

8 Variables

`\@@_parse_variables:n \@@_parse_variables:n {(integration variable)}, <key>`

This macro is applied to each element of the argument of the `variables` key to check whether it is a key or an explicit list, in the same way as for the limits. It mainly serves to transfer the elements of `\l_@@_variables_seq` (containing for example “ x, y, z ”) to `\l_@@_differential_seq` (which will contain “ dx, dy, dz ”); while taking into account the different options that have been applied. In the case where no variable is entered, the macro takes care of placing x or $\vec{r}$ depending on the value of the boolean `\l_@@_vectorial_differential_bool`.

```

335 \cs_new_protected:Nn \__integragl_parse_variables:
336 {
337     \bool_if:NF \l__integragl_custom_variables_bool
338     {
339         \bool_if:NTF \l__integragl_vectorial_differential_bool
340         {
341             \seq_set_eq:NN
342             \l__integragl_variables_seq
343             \l__integragl_default_vectorial_variables_seq
344         }
345         {
346             \seq_set_eq:NN
347             \l__integragl_variables_seq
348             \l__integragl_default_variables_seq
349         }
350     }
351     \seq_map_indexed_inline:Nn \l__integragl_variables_seq
352     {
353         \seq_put_right:Ne \l__integragl_differential_seq
354         {
355             \exp_not:V \l__integragl_differential_symbol_tl
356             \clist_if_empty:NF \l__integragl_differential_order_clist
357             {
358                 \c_math_superscript_token
359                 {
360                     \clist_item:Nn
361                     \l__integragl_differential_order_clist
362                     { ##1 }
363                 }
364             }
365             \bool_if:NT \l__integragl_vectorial_differential_bool
366             { \exp_not:V \l__integragl_vectorial_differential_style_tl }
367             { ##2 }
368         }
369     }
370 }
```

9 Special syntax

`\@@_retrieve_key_name:w ★ \@@_retrieve_key_name:w <clé>=<valeur> \q_stop`

In order to handle the special syntax, we start by creating a delimited macro that extracts the key name and its value, and that only returns the former.

```
371 \cs_new:Npn \__intexgral_retrieve_key_name:w #1=#2\q_stop { #1 }
```

```
\@@_extract_first_key_name:n \@@_extract_first_key_name:n {<\integral's optional argument>}
```

We then associate the optional argument received by `\integral` to a temporary *comma list*, from which we extract the first element; we then check if it contains an “=” sign. If so, we use the previous macro to retrieve only the key name; otherwise, we simply copy the group of tokens retrieved into `\l_@@_key_name_tl`.

```
372 \cs_new_protected:Npn \__intexgral_extract_first_key_name:n #1
373 {
374   \seq_set_split:Nnn \l_tmpa_seq { : } { #1 }
375   \tl_set:Nc \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
376   \tl_if_in:NnTF \l_tmpa_tl { = }
377   {
378     \tl_set:Nc \l__intexgral_key_name_tl
379     {
380       \exp_last_unbraced:NV
381       \__intexgral_retrieve_key_name:w \l_tmpa_tl \q_stop
382     }
383   }
384   { \tl_set_eq:NN \l__intexgral_key_name_tl \l_tmpa_tl }
385 }
```

```
\@@_parse_integral_keys:n \@@_parse_integral_keys:n {<first key of the optional argument>}
```

Using the `l3keys` module, we can check if the captured group of tokens corresponds to a defined key, in which case we directly apply `\keys_set:nn`. If the test is negative, we manually assign the keys by considering the optional argument as a sequence to be delimited with a colon. Temporary sequences are used to manipulate the different parts of the argument. We notably check for the presence of the token `+j` to activate the `jacobian` key, and we add the default variable if it is not specified. Finally, we assign values to the `limits`, `variables` and `mode` keys based on the extracted elements.

```
386 \cs_new_protected:Npn \__intexgral_parse_integral_keys:n #1
387 {
388   \keys_if_exist:nVTF { integral } \l__intexgral_key_name_tl
389   { \keys_set:nn { integral } { #1 } }
390   {
391     \regex_split:nnN { : } { #1 } \l_tmpa_seq
392     \str_if_eq:eeTF
393     { \seq_item:Nn \l_tmpa_seq { 2 } }
394     { +j }
395     {
396       \exp_args:NNne \seq_set_item:Nnn \l_tmpa_seq { 2 }
397       {
398         \seq_use:Nn
399         \l__intexgral_default_variables_seq
400         { , }
401         +j
402       }
403     }
404     {
405       \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 2 }
406       {
407         \seq_put_right:Ne
408         \l_tmpa_seq
409         { \seq_use:Nn \l__intexgral_default_variables_seq { , } }
```

```

410     }
411 }
412 \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 3 }
413   { \seq_put_right:Nn \l_tmpa_seq { d } }
414 \seq_set_split:Nne \l_tmpb_seq
415   { + }
416   { \seq_item:Nn \l_tmpa_seq { 2 } }
417 \keys_set:ne { integral }
418 {
419   limits      = { \seq_item:Nn \l_tmpa_seq { 1 } },
420   variables   = { \seq_item:Nn \l_tmpb_seq { 1 } },
421   mode        =
422   {
423     \str_case:enF { \seq_item:Nn \l_tmpa_seq { 3 } }
424     {
425       { d      } { default }
426       { n      } { nested  }
427       { p      } { product  }
428       { default } { default  }
429       { nested } { nested   }
430       { product } { product   }
431     }
432     { default }
433   },
434   \bool_if:nT
435   {
436     \int_compare_p:nNn { \seq_count:N \l_tmpb_seq } > { 1 }
437     &&
438     \str_if_eq_p:en { \seq_item:Nn \l_tmpb_seq { 2 } } { j }
439   }
440   { jacobian }
441 }
442 }
443 }

```

10 Typesetting the integral

`\@@_integral_preconfiguration:`

Before being able to typeset the integral in the document, a number of checks must be performed. Indeed, depending on the combination of keys specified in the optional argument, the sequences may not be filled yet. We therefore perform the following checks:

- We check if the integrand should be split into several parts by the `nested` and `product` modes, in which case we populate the integrand sequence by splitting the elements at the semicolon. Otherwise, we simply add the integrand (`\l_@@_integrand_tl`) to the sequence.
- We verify if the sequence of symbols is empty, indicating that the `limits*` key has not been called. Indeed, only the use of this key triggers the execution of all macros presented in section 7. In this case, we simply add to the sequence the general form of the integral, whose symbol and bounds will be optionally modified by the keys `symbol`, `ulimit` and `llimit`.
- We then control the deactivation of differentials in addition to detecting the presence of `\differentials` using a regular expression.

```

444 \cs_new_protected:Nn \__intexgral_integral_preconfiguration:
445 {
446   \bool_if:NTF \l__intexgral_separate_integral_bool
447   {
448     \seq_set_split:NnV
449       \l__intexgral_integrand_seq
450       { ; }
451     \l__intexgral_integrand_tl
452   }
453   {
454     \seq_put_right:NV
455       \l__intexgral_integrand_seq
456       \l__intexgral_integrand_tl
457   }
458   \seq_if_empty:NT \l__intexgral_integral_symbol_seq
459   {
460     \seq_put_right:Nn \l__intexgral_integral_symbol_seq
461     { \__intexgral_general_integral_symbol: }
462   }
463   \bool_if:NF \l__intexgral_deactivate_variables_bool
464   { \__intexgral_parse_variables: }
465   \tl_if_regex_match:VnTF \l__intexgral_integrand_tl { \c{differentials} }
466   { \bool_set_true:N \l__intexgral_manual_differential_bool }
467   { \bool_set_false:N \l__intexgral_manual_differential_bool }
468   \tl_if_regex_match:VnTF \l__intexgral_integrand_tl { \c{jacobian} }
469   { \bool_set_true:N \l__intexgral_manual_jacobian_bool }
470   { \bool_set_false:N \l__intexgral_manual_jacobian_bool }
471 }

```

We can now define the six macros that will be used to typeset a precise integral; three modes allowed by `mode`, each accompanied by a variant for the inverted differentials. It should also be noted that `\differentials` is only created if the boolean `\l_@@_manual_differential_bool` is true, in order to avoid unnecessary definitions.

10.1 Mode *default*

`\@@_print_default_integral:`

For the default mode, we typeset in the following order: integral symbol, integrand, Jacobian (if applicable), and differentials. Since the elements are successive, we apply `\seq_use:Nn` for each of the sequences, inserting a kern when necessary. Note that if `\l_@@_vectorial_differential_bool` is true, we add a middle dot and remove the previous kern (for spacing reasons related to `\cdot`).

```
472 \cs_new_protected:Npn \__intexgral_print_default_integral:
473 {
474     \bool_if:NT \l__intexgral_manual_differential_bool
475     {
476         \cs_set_protected:Npn \differentials
477         {
478             \seq_use:Nn \l__intexgral_differential_seq
479             { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
480         }
481     }
482     \bool_if:nT
483     { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
484     {
485         \cs_set_protected:Npn \jacobian
486         {
487             \seq_use:Nn \l__intexgral_jacobian_seq
488             { \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip }
489         }
490     }
491     \seq_use:Nn \l__intexgral_integral_symbol_seq
492     { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
493     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
494     \seq_use:Nn \l__intexgral_integrand_seq { }
495     \bool_if:nT
496     { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
497     {
498         \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
499         \seq_use:Nn \l__intexgral_jacobian_seq { }
500     }
501     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
502     \bool_if:NT \l__intexgral_vectorial_differential_bool
503     { \tex_unkern:D \cdot }
504     \bool_if:NF \l__intexgral_manual_differential_bool
505     {
506         \seq_use:Nn \l__intexgral_differential_seq
507         { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
508     }
509 }
```

`\@@_print_default_integral_inv:`

For the inverted differentials, nothing is very different, except that `\l_@@_integrand_seq` and `\l_@@_differential_seq` are swapped.

```
510 \cs_new_protected:Npn \__intexgral_print_default_integral_inv:
511 {
512     \bool_if:NT \l__intexgral_manual_differential_bool
513     {
```

```

514         \cs_set_protected:Npn \differentials
515         {
516             \seq_use:Nn \l__intexgral_differential_seq
517             { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
518         }
519     }
520 \bool_if:nT
521 { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
522 {
523     \cs_set_protected:Npn \jacobian
524     {
525         \seq_use:Nn \l__intexgral_jacobian_seq
526         { \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip }
527     }
528 }
529 \seq_use:Nn \l__intexgral_integral_symbol_seq
530 { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
531 \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
532 \bool_if:NF \l__intexgral_manual_differential_bool
533 {
534     \seq_use:Nn \l__intexgral_differential_seq
535     { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
536 }
537 \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
538 \bool_if:NT \l__intexgral_vectorial_differential_bool
539 { \tex_unkern:D \cdot}
540 \seq_use:Nn \l__intexgral_integrand_seq { }
541 \bool_if:nT { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
542 {
543     \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
544     \seq_use:Nn \l__intexgral_jacobian_seq { }
545 }
546 }

```

10.2 Mode *nested*

\@@_print_nested_integral:

In the nested mode, we initiate an incremented loop where we extract the i^{th} element of each of the sequences of symbol, integrand and Jacobian. After the iteration, we compose all the differentials again using `\seq_use:Nn`.

```

547 \cs_new_protected:Npn \__intexgral_print_nested_integral:
548 {
549     \bool_if:NT \l__intexgral_manual_differential_bool
550     {
551         \cs_set_protected:Npn \differentials
552         {
553             \seq_use:Nn \l__intexgral_differential_seq
554             { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
555         }
556     }
557 \bool_if:nT
558 { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
559 {
560     \cs_set_protected:Npn \jacobian
561     {

```

```

562         \seq_gpop_left:NNT \l__intexgral_jacobian_seq \l_tmpa_tl
563         { \tl_use:N \l_tmpa_tl }
564     }
565 }
566 \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
567 {
568     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
569     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
570     \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
571     \bool_if:nT
572     { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
573     {
574         \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
575         \seq_item:Nn \l__intexgral_jacobian_seq { ##1 }
576     }
577 }
578 \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
579 \bool_if:NF \l__intexgral_manual_differential_bool
580 {
581     \seq_use:Nn \l__intexgral_differential_seq
582     { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
583 }
584 }

```

\@@_print_nested_integral_inv:

Here, it is the sequence of differentials that appears in the loop and the sequence of the integrand that is used at the end, after the iteration.

```

585 \cs_new_protected:Npn \__intexgral_print_nested_integral_inv:
586 {
587     \bool_if:nT
588     { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
589     {
590         \cs_set_protected:Npn \jacobian
591         {
592             \seq_use:Nn \l__intexgral_jacobian_seq
593             { \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip }
594         }
595     }
596     \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
597     {
598         \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
599         \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
600         \seq_item:Nn \l__intexgral_differential_seq { ##1 }
601     }
602     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
603     \seq_use:Nn \l__intexgral_integrand_seq { }
604     \bool_if:nT { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
605     {
606         \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
607         \seq_use:Nn \l__intexgral_jacobian_seq { }
608     }
609 }

```

10.3 Mode *product*

\@@_print_product_integral:

The product mode is very similar to the nested mode, in the sense that all sequences appear in the loop. Unlike the two previous modes, `\differentials` is defined differently this time: it extracts the elements of `\l_@@_differential_seq` in order and then displays them. The extraction is global in case the macro is used within a group.

```

610 \cs_new_protected:Npn \__intexgral_print_product_integral:
611 {
612   \bool_if:NT \l__intexgral_manual_differential_bool
613   {
614     \cs_set_protected:Npn \differentials
615     {
616       \seq_gpop_left:NNT \l__intexgral_differential_seq \l_tmpa_tl
617       { \tl_use:N \l_tmpa_tl }
618     }
619   }
620   \bool_if:nT
621   { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
622   {
623     \cs_set_protected:Npn \jacobian
624     {
625       \seq_gpop_left:NNT \l__intexgral_jacobian_seq \l_tmpa_tl
626       { \tl_use:N \l_tmpa_tl }
627     }
628   }
629   \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
630   {
631     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
632     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
633     \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
634     \bool_if:nT
635     { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
636     {
637       \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
638       \seq_item:Nn \l__intexgral_jacobian_seq { ##1 }
639     }
640     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
641     \bool_if:NF \l__intexgral_manual_differential_bool
642     { \seq_item:Nn \l__intexgral_differential_seq { ##1 } }
643   }
644 }

```

\@@_print_product_integral_inv:

Similarly, with a few adjustments.

```

645 \cs_new_protected:Npn \__intexgral_print_product_integral_inv:
646 {
647   \bool_if:NT \l__intexgral_manual_differential_bool
648   {
649     \cs_set_protected:Npn \differentials
650     {
651       \seq_gpop_left:NNT \l__intexgral_differential_seq \l_tmpa_tl
652       { \tl_use:N \l_tmpa_tl }
653     }

```

```

654     }
655     \bool_if:nT
656     { \l__intexgral_jacobian_bool && \l__intexgral_manual_jacobian_bool }
657     {
658         \cs_set_protected:Npn \jacobian
659         {
660             \seq_gpop_left:NNT \l__intexgral_jacobian_seq \l_tmpa_tl
661             { \tl_use:N \l_tmpa_tl }
662         }
663     }
664     \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
665     {
666         \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
667         \bool_if:NF \l__intexgral_manual_differential_bool
668         {
669             \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
670             \seq_item:Nn \l__intexgral_differential_seq { ##1 }
671         }
672         \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
673         \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
674         \bool_if:nT
675         { \l__intexgral_jacobian_bool && !\l__intexgral_manual_jacobian_bool }
676         {
677             \__intexgral_mkern:N \l__intexgral_jacobian_sep_muskip
678             \seq_item:Nn \l__intexgral_jacobian_seq { ##1 }
679         }
680     }
681 }

```

10.4 Final typesetting

\@@_print_integral:

The main macro typesetting the integral first increments the global counter allowing each integral to be numbered. Then, it carries out the preparatory actions explained previously. Finally, depending on the placement of the differentials and the requested display mode, one of the six macros is called.

```

682 \cs_new_protected:Nn \__intexgral_print_integral:
683 {
684     \int_gincr:N \g__intexgral_integral_number_int
685     \__intexgral_integral_preconfiguration:
686     \cs_set:Npn \variables
687     { \seq_use:Nn \l__intexgral_variables_seq { , } }
688     \bool_if:NTF \l__intexgral_invert_differential_bool
689     {
690         \str_case:VnF \l__intexgral_display_mode_str
691         {
692             { default } { \__intexgral_print_default_integral_inv: }
693             { nested } { \__intexgral_print_nested_integral_inv: }
694             { product } { \__intexgral_print_product_integral_inv: }
695         }
696         { \__intexgral_print_default_integral_inv: }
697     }
698     {
699         \str_case:VnF \l__intexgral_display_mode_str
700         {

```

```

701         { default } { \__intexgral_print_default_integral: }
702         { nested } { \__intexgral_print_nested_integral: }
703         { product } { \__intexgral_print_product_integral: }
704     }
705     { \__intexgral_print_default_integral: }
706 }
707 }

```

11 Key declaration

The creation of keys for `\integral` and `\IntegralSetup` is relatively trivial. The few elementary actions for the keys `limits` and `variables` that were not covered by the macros are performed here.

```

708 \keys_define:nn { integral }
709 {
710     mode .choice:,
711     mode / default .code:n =
712     {
713         \bool_set_false:N \l__intexgral_separate_integral_bool
714         \str_set:Nn \l__intexgral_display_mode_str { default }
715     },
716     mode / nested .code:n =
717     {
718         \bool_set_true:N \l__intexgral_separate_integral_bool
719         \str_set:Nn \l__intexgral_display_mode_str { nested }
720     },
721     mode / product .code:n =
722     {
723         \bool_set_true:N \l__intexgral_separate_integral_bool
724         \str_set:Nn \l__intexgral_display_mode_str { product }
725     },
726     mode .default:n = { default },
727
728     limits-mode .choice:,
729     limits-mode / limits .code:n =
730     { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_limits:D },
731     limits-mode / nolimits .code:n =
732     { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_nolimits:D },
733
734     limits .code:n =
735     {
736         \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
737         \__intexgral_set_limits_regular:
738         \__intexgral_generate_integral_sequence:
739     },
740
741     limits* .code:n =
742     {
743         \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
744         \__intexgral_set_limits_starred:
745         \__intexgral_generate_integral_sequence:
746     },
747
748     llimit .tl_set:N = \l__intexgral_lower_limit_tl,
749
750     ulimit .tl_set:N = \l__intexgral_upper_limit_tl,

```

```

751
752 symbol .code:n =
753 {
754     \cs_if_exist:NTF #1
755     { \tl_set_eq:NN \l__intexgral_integral_symbol_tl #1 }
756     {
757         \msg_warning:nnn { intexgral } { unknown-symb } { #1 }
758         \tl_set_eq:NN \l__intexgral_integral_symbol_tl \int
759     }
760 },
761
762 nint .code:n =
763 {
764     \int_compare:nNnT { #1 } < { 5 }
765     { \msg_warning:nn { intexgral } { use-glyph-instead } }
766     \tl_clear:N \l__intexgral_integral_symbol_tl
767     \int_step_inline:nn { #1 }
768     {
769         \tl_put_right:Nn \l__intexgral_integral_symbol_tl
770         { \int }
771         \int_compare:nNnT { ##1 } < { #1 }
772         {
773             \tl_put_right:Nn \l__intexgral_integral_symbol_tl
774             {
775                 \__intexgral_mathchoice:nnnn
776                 { \tex_mkern:D -12mu \scan_stop: }
777                 { \tex_mkern:D -8mu \scan_stop: }
778                 { \tex_mkern:D -4mu \scan_stop: }
779                 { \tex_mkern:D -2mu \scan_stop: }
780             }
781         }
782     }
783 },
784
785 domain .code:n =
786 {
787     \tl_set:Nn \l_tmpa_tl { #1 }
788     \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
789     \seq_map_inline:Nn \l__intexgral_domain_seq
790     {
791         \tl_if_empty:NF \l__intexgral_lower_limit_tl
792         { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
793         \tl_set:Ne \l__intexgral_domain_char_tl
794         { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
795         \tl_set:Ne \l__intexgral_domain_dimension_tl
796         { \tl_tail:n { ##1 } }
797         \tl_put_right:Ne \l__intexgral_lower_limit_tl
798         {
799             \exp_not:V \l__intexgral_domain_style_tl
800             { \l__intexgral_domain_char_tl }
801             \c_math_superscript_token { \l__intexgral_domain_dimension_tl }
802         }
803     }
804 },
805
806 domain* .code:n =
807 {
808     \tl_set:Nn \l_tmpa_tl { #1 }

```

```

809 \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
810 \seq_map_inline:Nn \l__intexgral_domain_seq
811 {
812   \tl_if_empty:NF \l__intexgral_lower_limit_tl
813   { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
814   \tl_set:Ne \l__intexgral_domain_char_tl
815   { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
816   \tl_set:Ne \l__intexgral_domain_dimension_tl
817   { \tl_tail:n { ##1 } }
818   \tl_put_right:Ne \l__intexgral_lower_limit_tl
819   {
820     \exp_not:V \l__intexgral_domain_style_tl
821     { \l__intexgral_domain_char_tl }
822     \c_math_subscript_token { \l__intexgral_domain_dimension_tl }
823   }
824 }
825 },
826
827 boundary .code:n =
828   { \tl_set:Nn \l__intexgral_lower_limit_tl { \partial #1 } },
829
830 variables .code:n =
831   {
832     \bool_set_true:N \l__intexgral_custom_variables_bool
833     \str_if_eq:nnTF { #1 } { none }
834     { \bool_set_true:N \l__intexgral_deactivate_variables_bool }
835     {
836       \prop_get:NnNTF
837       \g__intexgral_differential_group_keyword_prop { #1 } \l_tmpa_tl
838       {
839         \seq_set_split:NnV
840         \l__intexgral_variables_seq
841         { , }
842         \l_tmpa_tl
843         \seq_if_exist:cTF { g__intexgral_#1_jacobian_seq }
844         {
845           \seq_set_eq:Nc
846           \l__intexgral_jacobian_seq
847           { g__intexgral_#1_jacobian_seq }
848         }
849         { \msg_warning:nnn { intexgral } { no-jacobian } { #1 } }
850       }
851       { \seq_set_split:Nnn \l__intexgral_variables_seq { , } { #1 } }
852     }
853   },
854
855 jacobian .code:n =
856   { \bool_set_true:N \l__intexgral_jacobian_bool },
857 diff-vec .code:n =
858   { \bool_set_true:N \l__intexgral_vectorial_differential_bool },
859 diff-symb .tl_set:N = \l__intexgral_differential_symbol_tl,
860 diff-order .clist_set:N = \l__intexgral_differential_order_clist,
861 }
862
863 \keys_define:nn { IntegralSetup }
864 {
865   diff-symb .tl_set:N = \l__intexgral_differential_symbol_tl,
866   defaultvar .code:n =

```

```

867 {
868   \str_if_eq:nnTF { #1 } { none }
869   { \bool_set_true:N \l__intexgral_deactivate_variables_bool }
870   {
871     \seq_set_split:Nne
872       \l__intexgral_default_variables_seq
873       { , }
874       {
875         \prop_if_in:NnTF \g__intexgral_differential_group_keyword_prop { #1
876           { \prop_item:Nn \g__intexgral_differential_group_keyword_prop { #
877             { #1 }
878           }
879         \seq_if_exist:cT { g__intexgral_#1_jacobian_seq }
880         {
881           \seq_set_eq:Nc
882             \l__intexgral_jacobian_seq
883             { g__intexgral_#1_jacobian_seq }
884         }
885       }
886   },
887   defaultvar* .code:n =
888   {
889     \str_if_eq:nnTF { #1 } { none }
890     { \bool_set_true:N \l__intexgral_deactivate_variables_bool }
891     {
892       \seq_set_split:Nne
893         \l__intexgral_default_vectorial_variables_seq
894         { , }
895         {
896           \prop_if_in:NnTF \g__intexgral_differential_group_keyword_prop { #1
897             { \prop_item:Nn \g__intexgral_differential_group_keyword_prop { #
898               { #1 }
899             }
900           \seq_if_exist:cT { g__intexgral_#1_jacobian_seq }
901           {
902             \seq_set_eq:Nc
903               \l__intexgral_jacobian_seq
904               { g__intexgral_#1_jacobian_seq }
905           }
906         }
907     },
908     postsymbsep .muskip_set:N = \l__intexgral_symbol_post_sep_muskip,
909     varsep .muskip_set:N = \l__intexgral_variable_sep_muskip,
910     diffsep .muskip_set:N = \l__intexgral_differential_sep_muskip,
911     jacobiansep .muskip_set:N = \l__intexgral_jacobian_sep_muskip,
912     innersymbsep .muskip_set:N = \l__intexgral_symbol_inner_sep_muskip,
913     vectorstyle .tl_set:N = \l__intexgral_vectorial_differential_style_tl,
914     domainstyle .tl_set:N = \l__intexgral_domain_style_tl,
915     novar .code:n =
916     {
917       \msg_warning:nn { intexgral } { novar-deprecated }
918       \bool_set_true:N \l__intexgral_deactivate_variables_bool,
919     },
920     delim .choice:,
921     delim / brackets .code:n =
922     {
923       \tl_set_eq:NN \l__intexgral_primitive_left_delim_tl \c__intexgral_left_bracke
924       \tl_set_eq:NN \l__intexgral_primitive_right_delim_tl \c__intexgral_right_brac

```

```

925     },
926     delim / bar .code:n =
927     {
928         \tl_set_eq:NN \l__intexgral_primitive_left_delim_tl .
929         \tl_set_eq:NN \l__intexgral_primitive_right_delim_tl \vert
930     },
931 }
932
933 \keys_define:nn { primitive }
934 {
935     limits .code:n =
936     {
937         \seq_set_split:Nnf \l_tmpa_seq { , } { \__intexgral_parse_limits:n { #1 } }
938         \tl_set:Ne \l__intexgral_lower_limit_tl
939         { \seq_item:Nn \l_tmpa_seq { 1 } }
940         \tl_set:Ne \l__intexgral_upper_limit_tl
941         { \seq_item:Nn \l_tmpa_seq { 2 } }
942     },
943     auto .code:n =
944     {
945         \tl_set_eq:NN \l__intexgral_primitive_left_delim_size_tl \left
946         \tl_set_eq:NN \l__intexgral_primitive_right_delim_size_tl \right
947     },
948     big .code:n =
949     {
950         \tl_set_eq:NN \l__intexgral_primitive_left_delim_size_tl \bigl
951         \tl_set_eq:NN \l__intexgral_primitive_right_delim_size_tl \bigr
952     },
953     Big .code:n =
954     {
955         \tl_set_eq:NN \l__intexgral_primitive_left_delim_size_tl \Bigl
956         \tl_set_eq:NN \l__intexgral_primitive_right_delim_size_tl \Bigr
957     },
958     bigg .code:n =
959     {
960         \tl_set_eq:NN \l__intexgral_primitive_left_delim_size_tl \biggl
961         \tl_set_eq:NN \l__intexgral_primitive_right_delim_size_tl \biggr
962     },
963     Bigg .code:n =
964     {
965         \tl_set_eq:NN \l__intexgral_primitive_left_delim_size_tl \Biggl
966         \tl_set_eq:NN \l__intexgral_primitive_right_delim_size_tl \Biggr
967     },
968     delim .choice:,
969     delim / brackets .code:n =
970     {
971         \tl_set_eq:NN \l__intexgral_primitive_left_delim_tl \c__intexgral_left_bracke
972         \tl_set_eq:NN \l__intexgral_primitive_right_delim_tl \c__intexgral_right_brac
973     },
974     delim / bar .code:n =
975     {
976         \tl_set_eq:NN \l__intexgral_primitive_left_delim_tl .
977         \tl_set_eq:NN \l__intexgral_primitive_right_delim_tl \vert
978     },
979 }

```

12 User-level macros

12.1 Keywords for limits

`\@@_new_limits_group:nn` `\@@_new_limits_group:nn` `{\keyword}` `{\limits}`

For the functioning of `\NewLimitsKeyword`, we register the keyword and its value in a property list. A particularity that nevertheless needs to be explained is that there is a double inversion of the limits: at the time of the creation of the keyword, and at the time of extracting its content (cf. `\@@_parse_limits:nn`). The reason is that it is difficult to attach to a keyword the status of *inverted limits*. Thus, whatever convention is chosen, the limits are stored *in the same order*. This also allows the keywords defined by the package to be in the correct order, even if the convention is changed after loading it.

```
980 \cs_new_protected:Npn \__intexgral_new_limits_group:nn #1#2
981   {
982     \prop_gput:Nne \g__intexgral_limits_keyword_prop { #1 }
983     {
984       \bool_if:NTF \l__intexgral_invert_limits_bool
985         { \__intexgral_invert_limits:w #2\q_stop }
986         { #2 }
987     }
988   }
```

`\NewLimitsKeyword`

This macro is documented in section 4.4.1.

```
989 \NewDocumentCommand\NewLimitsKeyword{ m m }
990   {
991     \prop_if_in:NnTF \g__intexgral_limits_keyword_prop { #1 }
992       { \msg_error:nnn { intexgral } { lim-group-alr-def } { #1 } }
993       { \__intexgral_new_limits_group:nn { #1 } { #2 } }
994   }
```

`\RenewLimitsKeyword`

This macro is documented in section 4.4.1.

```
995 \NewDocumentCommand\RenewLimitsKeyword{ m m }
996   {
997     \prop_pop:NnNTF \g__intexgral_limits_keyword_prop { #1 } \l_tmpa_tl
998       { \__intexgral_new_limits_group:nn { #1 } { #2 } }
999       { \msg_error:nnn { intexgral } { lim-group-not-def } { #1 } }
1000   }
```

`\ProvideLimitsKeyword`

This macro is documented in section 4.4.1.

```
1001 \NewDocumentCommand\ProvideLimitsKeyword{ m m }
1002   {
1003     \prop_if_in:NnF \g__intexgral_limits_keyword_prop { #1 }
1004       { \__intexgral_new_limits_group:nn { #1 } { #2 } }
1005   }
```

`\DeclareLimitsKeyword`

This macro is documented in section 4.4.1.

```
1006 \NewDocumentCommand\DeclareLimitsKeyword{ m m }
1007 {
1008   \prop_remove:Nn \g__intexgral_limits_keyword_prop { #1 }
1009   \__intexgral_new_limits_group:nn { #1 } { #2 }
1010 }
```

12.2 Keywords for variables

`\@@_new_variables_group:nnn \@@_new_variables_group:nnn {\keyword} {\variable} [<jacobian>]`

Just like for the limits, we register the groups of differentials in a property list, with their name, value and eventually their associated jacobian.

```
1011 \cs_new_protected:Npn \__intexgral_new_variables_group:nnn #1#2#3
1012 {
1013   \prop_gput:Nnn \g__intexgral_differential_group_keyword_prop
1014     { #1 } { #2 }
1015   \tl_if_blank:nF { #3 }
1016     {
1017       \seq_if_exist:cF { g__intexgral_#1_jacobian_seq }
1018       { \seq_new:c { g__intexgral_#1_jacobian_seq } }
1019       \seq_gset_split:cnn { g__intexgral_#1_jacobian_seq } { , } { #3 }
1020     }
1021 }
```

`\NewVariableKeyword`

This macro is documented in section 4.5.1.

```
1022 \NewDocumentCommand\NewVariableKeyword{ m m o }
1023 {
1024   \prop_if_in:NnTF \g__intexgral_differential_group_keyword_prop { #1 }
1025     { \msg_error:nnn { intexgral } { diff-group-alr-def } { #1 } }
1026     { \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 } }
1027 }
```

`\RenewVariableKeyword`

This macro is documented in section 4.5.1.

```
1028 \NewDocumentCommand\RenewVariableKeyword{ m m o }
1029 {
1030   \prop_pop:NnNTF \g__intexgral_differential_group_keyword_prop { #1 }
1031     \l_tmpa_tl
1032     {
1033       \seq_gclear:c { g__intexgral_#1_jacobian_seq }
1034       \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 }
1035     }
1036     { \msg_error:nnn { intexgral } { diff-group-not-def } { #1 } }
1037 }
```

`\ProvideVariableKeyword`

This macro is documented in section 4.5.1.

```

1038 \NewDocumentCommand\ProvideVariableKeyword{ m m o }
1039 {
1040     \prop_if_in:NnF \g__integragl_differential_group_keyword_prop { #1 }
1041     { \__integragl_new_variables_group:nnn { #1 } { #2 } { #3 } }
1042 }

```

\DeclareVariableKeyword

This macro is documented in section 4.5.1.

```

1043 \NewDocumentCommand\DeclareVariableKeyword{ m m o }
1044 {
1045     \prop_remove:Nn \g__integragl_differential_group_keyword_prop { #1 }
1046     \seq_gclear:c { g__integragl_#1_jacobian_seq }
1047     \__integragl_new_variables_group:nnn { #1 } { #2 } { #3 }
1048 }

```

12.3 Keywords for symbols

This section does not require an intermediate macro, since we modify the keys of the *integral* module directly.

\NewSymbolKeyword

This macro is documented in section 4.6.

```

1049 \NewDocumentCommand\NewSymbolKeyword{ m m }
1050 {
1051     \prop_if_in:NnT \g__integragl_limits_keyword_prop { #1 }
1052     { \msg_warning:nnn { integragl } { key-exists-for-lim } { #1 } }
1053     \keys_if_exist:nnTF { integral } { #1 }
1054     { \msg_error:nnn { integragl } { symb-key-alr-def } { #1 } }
1055     {
1056         \keys_define:nn { integral }
1057         {
1058             #1 .meta:n =
1059             {
1060                 symbol=#2,
1061                 llimit=##1
1062             },
1063         }
1064     }
1065 }

```

\RenewSymbolKeyword

This macro is documented in section 4.6.

```

1066 \NewDocumentCommand\RenewSymbolKeyword{ m m }
1067 {
1068     \prop_if_in:NnT \g__integragl_limits_keyword_prop { #1 }
1069     { \msg_warning:nnn { integragl } { key-exists-for-lim } { #1 } }
1070     \keys_if_exist:nnTF { integral } { #1 }
1071     {
1072         \keys_define:nn { integral }
1073         {
1074             #1 .undefine:,
1075             #1 .meta:n =
1076             {

```

```

1077             symbol=#2,
1078             llimit=##1
1079         }
1080     }
1081 }
1082 { \msg_error:nnn { intexgral } { symb-key-not-def } }
1083 }

```

\ProvideSymbolKeyword

This macro is documented in section 4.6.

```

1084 \NewDocumentCommand\ProvideSymbolKeyword{ m m }
1085 {
1086     \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
1087     { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
1088     \keys_if_exist:nnF { integral } { #1 }
1089     {
1090         \keys_define:nn { integral }
1091         {
1092             #1 .meta:n =
1093             {
1094                 symbol=#2,
1095                 llimit=##1
1096             },
1097         }
1098     }
1099 }

```

\DeclareSymbolKeyword

This macro is documented in section 4.6.

```

1100 \NewDocumentCommand\DeclareSymbolKeyword{ m m }
1101 {
1102     \keys_define:nn { integral }
1103     {
1104         #1 .undefine:,
1105         #1 .meta:n =
1106         {
1107             symbol=#2,
1108             llimit=##1
1109         },
1110     }
1111 }

```

12.4 Other macros

\IntegralSetup

This macro is documented in section 6.

```

1112 \NewDocumentCommand\IntegralSetup{ m }
1113 { \keys_set:nn { IntegralSetup } { #1 } }

```

\intexgralsetup

Just like the previous macro, but without forgetting to restrict the usage to the preamble.

This macro is documented in section 1.

```

1114 \NewDocumentCommand\intexgralsetup{ m }
1115   { \keys_set:nn { intexgral } { #1 } }
1116 \@onlypreamble\intexgralsetup

```

\invertdiff

Just a trivial definition that simply inverts the boolean seen at the beginning of the implementation.

```

1117 \NewDocumentCommand\invertdiff{ }
1118   { \bool_set_inverse:N \l__intexgral_invert_differential_bool }

```

\differentials \jacobian

We define these three macros such that their use outside of `\integral` triggers an error message.

```

1119 \cs_new_protected:Npn \differentials
1120   { \msg_error:nn { intexgral } { differentials-outside } }
1121 \cs_new_protected:Npn \jacobian
1122   { \msg_error:nn { intexgral } { jacobian-outside } }
1123 \cs_new_protected:Npn \variables
1124   { \msg_error:nn { intexgral } { variables-outside } }

```

\@@_print_primitive:

This macro is used by `\antideriv` to typeset the antiderivative.

```

1125 \cs_new_protected:Nn \__intexgral_print_primitive:
1126   {
1127     \bool_lazy_and:nnT
1128       { \tl_if_empty_p:N \l__intexgral_primitive_left_delim_size_tl }
1129       { \tl_if_eq_p:NN \l__intexgral_primitive_left_delim_tl . }
1130       { \use_none:nn }
1131     \l__intexgral_primitive_left_delim_size_tl
1132     \l__intexgral_primitive_left_delim_tl
1133     \l__intexgral_primitive_tl
1134     \l__intexgral_primitive_right_delim_size_tl
1135     \l__intexgral_primitive_right_delim_tl
1136     \c_math_subscript_token { \l__intexgral_lower_limit_tl }
1137     \c_math_superscript_token { \l__intexgral_upper_limit_tl }
1138   }

```

\antideriv

```

1139 \NewDocumentCommand\antideriv{ 0{} m }
1140   {
1141     \group_begin:
1142     \keys_set:nn { primitive } { #1 }
1143     \tl_set:Nn \l__intexgral_primitive_tl { #2 }
1144     \__intexgral_print_primitive:
1145     \group_end:
1146   }

```

\integral

The final macro is only a few lines long. If an optional argument is provided, it performs the necessary checks to determine the nature of that argument (whether it follows special syntax or not) and assigns the keys accordingly. It then assigns the main argument to the corresponding token before calling the central macro that constructs the entire expression. The entire process is executed within a group so that all modifications remain local.

This macro is documented in section 2.

```

1147 \NewDocumentCommand\integral{ 0{} m }
1148 {
1149   \group_begin:
1150   \tl_if_empty:nF { #1 }
1151   {
1152     \__intexgral_extract_first_key_name:n { #1 }
1153     \__intexgral_parse_integral_keys:n { #1 }
1154   }
1155   \tl_set:Nn \l__intexgral_integrand_tl { #2 }
1156   \__intexgral_print_integral:
1157   \group_end:
1158 }

```

13 Package configuration

The last part of the package is dedicated to defining the default keywords for symbols, limits and variables, as well as the default parameters for the typesetting of integrals.

13.1 Symbol

```

1159 \NewSymbolKeyword{single}      {\int}
1160 \NewSymbolKeyword{double}     {\iint}
1161 \NewSymbolKeyword{triple}     {\iiint}
1162 \NewSymbolKeyword{quadruple}  {\iiiiint}
1163 \NewSymbolKeyword{contour}    {\oint}
1164 \NewSymbolKeyword{surface}    {\oiint}
1165 \NewSymbolKeyword{volume}     {\oiiint}

```

13.2 Limits

```

1166 \NewLimitsKeyword{ab}         {a, b}
1167 \NewLimitsKeyword{real}       {-\infty, +\infty}
1168 \NewLimitsKeyword{positive}   {0, +\infty}
1169 \NewLimitsKeyword{negative}   {-\infty, 0}
1170 \NewLimitsKeyword{unit}       {0, 1}
1171 \NewLimitsKeyword{circle}     {0, 2\pi}
1172 \NewLimitsKeyword{scircle}    {0, \pi}
1173 \NewLimitsKeyword{qcircle}    {0, \pi/2}
1174 \NewLimitsKeyword{height}     {0, H}
1175 \NewLimitsKeyword{radius}     {0, R}
1176 \NewLimitsKeyword{length}     {0, L}
1177 \NewLimitsKeyword{time}       {0, T}

```

13.3 Variables

```

1178 \NewVariableKeyword{cartesian} {x, y, z}
1179 \NewVariableKeyword{planar}    {x, y}
1180 \NewVariableKeyword{polar}     {r, \theta} [r]
1181 \NewVariableKeyword{cylindrical} {r, \theta, z} [r]
1182 \NewVariableKeyword{cylindrical*} {\rho, \theta, z} [\rho]
1183 \NewVariableKeyword{spherical} {r, \theta, \phi} [r^2, \sin\theta]

```

13.4 Default parameters

```

1184 \IntegralSetup{
1185   diff-symb    = \l__intexgral_default_differential_symbol_tl,
1186   defaultvar   = {x},

```

```

1187   defaultvar* = {r},
1188   varsep      = \thinmuskip,
1189   diffsep     = \thinmuskip,
1190   innersymbsep = {-5mu},
1191   postsymbsep  = {-1mu},
1192   jacobiansep  = {0mu},
1193   vectorstyle  = \vec,
1194   domainstyle  = \mathbb,
1195   delim        = brackets,
1196 }
1197 \end{package}

```

Index

Symbols

_ 93, 97, 101, 105, 109, 113, 118

A

\antideriv 1139

B

\Biggl 965

\biggl 960

\Biggr 966

\biggr 961

\Bigl 955

\bigl 950

\Bigr 956

\bigr 951

bool commands:

\bool_case:nTF 237

\bool_if:NTF
. 241, 275, 292, 308, 337,
339, 365, 446, 463, 474, 502,
504, 512, 532, 538, 549, 579,
612, 641, 647, 667, 688, 984

\bool_if:nTF
. 434, 482, 495, 520, 541, 557,
571, 587, 604, 620, 634, 655,
674

\bool_lazy_and:nnTF 228, 1127

\bool_new:N 28, 29, 30, 184, 185,
186, 187, 188, 189

\bool_set_false:N 467, 470, 713

\bool_set_inverse:N 1118

\bool_set_true:N 466, 469, 718,
723, 832, 834, 856, 858, 869,
890, 918

C

\c 465, 468

\cdot 503, 539

clist commands:

\clist_if_empty:NTF 356

\clist_item:Nn 360

\clist_new:N 193

cs commands:

\cs_generate_variant:Nn . 158,
159, 160, 161, 162, 163, 164, 165

\cs_if_exist:NTF 27, 754

\cs_new:Nn 83

\cs_new:Npn . 205, 206, 234, 235,
371

\cs_new_eq:NN 204

\cs_new_protected:Nn . 257, 318,
335, 444, 682, 1125

\cs_new_protected:Npn . . . 202,
269, 306, 372, 386, 472, 510,
547, 585, 610, 645, 980, 1011,
1119, 1121, 1123

\cs_set:Npn 686

\cs_set_protected:Npn
. 476, 485, 514, 523, 551, 560,
590, 614, 623, 649, 658

\cs_to_str:N 145, 148, 151

D

\DeclareLimitsKeyword 1006

\DeclareSymbolKeyword 1100

\DeclareVariableKeyword . . . 1043

\def 6, 7, 8, 9

\differentials 145, 476, 514, 551, 614,
649, 1119

E

exp commands:

\exp_args:Ne 229, 230, 794, 815

\exp_args:NNne 396

\exp_last_unbraced:Ne . . . 243

\exp_last_unbraced:NV . . . 380

\exp_not:n 310, 311, 314, 315, 355,
366, 799, 820

G

group commands:

\group_begin: 1141, 1149

\group_end: 1145, 1157

H

hook commands:

\hook_gput_code:nnn 26

I

\iiiint 1162

\iiint 1161

\iint 1160

\infty 297, 301, 1167, 1168, 1169

\int 127, 168, 758, 770, 1159

int commands:

\int_compare:nNnTF . . 405, 412,
764, 771

\int_compare_p:nNn 436

\int_gincr:N 684

\int_gzero_new:N 82

\int_set:Nn 320

`\int_step_inline:nn` 326, 566, 596, 629, 664, 767
`\int_use:N` 86
`\l_tmpa_int` 320, 326
`\integral` 1147
`\IntegralSetup` 1112, 1184
integral internal commands:
`\l__intexgral_custom_variables_-bool` 186, 337, 832
`\l__intexgral_deactivate_variables_bool` 29, 463, 834, 869, 890, 918
`\l__intexgral_default_differential_symbol_tl` . 31, 55, 60, 69, 74, 1185
`\l__intexgral_default_variables_seq` 191, 348, 399, 409, 872
`\l__intexgral_default_vectorial_variables_seq` ... 192, 343, 893
`\g__intexgral_differential_group_keyword_prop` 195, 837, 875, 876, 896, 897, 1013, 1024, 1030, 1040, 1045
`\l__intexgral_differential_order_clist` . 193, 356, 361, 860
`\l__intexgral_differential_sep_muskip` 199, 479, 507, 517, 535, 554, 582, 910
`\l__intexgral_differential_seq` 221, 353, 478, 506, 516, 534, 553, 581, 600, 616, 642, 651, 670
`\l__intexgral_differential_symbol_tl` 171, 355, 859, 865
`\l__intexgral_display_mode_str` .. 201, 690, 699, 714, 719, 724
`\l__intexgral_domain_char_tl` .. 174, 793, 800, 814, 821
`\l__intexgral_domain_dimension_tl` 175, 795, 801, 816, 822
`\l__intexgral_domain_seq` 190, 788, 789, 809, 810
`\l__intexgral_domain_style_tl` 173, 799, 820, 914
`\__intexgral_extract_first_key_name:n` 372, 1152
`\__intexgral_general_integral_symbol:` 206, 332, 461
`\__intexgral_generate_integral_sequence:` 318, 738, 745
`\__intexgral_has_pm:n` ... 226
`\__intexgral_has_pm_p:n` . 249
`\g__intexgral_integral_number_int` 82, 86, 684
`\__intexgral_integral_preconfiguration:` 444, 685
`\l__intexgral_integral_symbol_seq` .. 218, 331, 458, 460, 491, 529, 566, 568, 596, 598, 629, 631, 664, 666
`\l__intexgral_integral_symbol_tl` 167, 168, 210, 755, 758, 766, 769, 773
`\l__intexgral_integrand_seq` . 219, 449, 455, 494, 540, 570, 603, 633, 673
`\l__intexgral_integrand_tl` .. 166, 451, 456, 465, 468, 1155
`\l__intexgral_invert_differential_bool` 30, 39, 688, 1118
`\__intexgral_invert_limits:w` 205, 243, 985
`\l__intexgral_invert_limits_bool` 28, 35, 241, 275, 292, 308, 984
`\l__intexgral_jacobian_bool` .. 189, 483, 496, 521, 541, 558, 572, 588, 604, 621, 635, 656, 675, 856
`\l__intexgral_jacobian_sep_muskip` 200, 488, 498, 526, 543, 574, 593, 606, 637, 677, 911
`\l__intexgral_jacobian_seq` .. 220, 487, 499, 525, 544, 562, 575, 592, 607, 625, 638, 660, 678, 846, 882, 903
`\l__intexgral_key_name_tl` 178, 378, 384, 388
`\c__intexgral_left_bracket_tl` .. 176, 298, 301, 923, 971

`\g__intexgral_limits_keyword_` -
`prop` 194, 239, 244, 247,
982, 991, 997, 1003, 1008, 1051,
1068, 1086
`\l__intexgral_limits_seq` 223,
259, 271, 736, 743
`\l__intexgral_limits_style_tl`
. . 32, 45, 47, 211, 730, 732
`\l__intexgral_lower_limit_tl`
. . 169, 213, 277, 285, 299, 310,
314, 748, 791, 792, 797, 812, 813,
818, 828, 938, 1136
`\l__intexgral_lower_limits_` -
`seq` . 225, 263, 294, 322, 324,
329
`\l__intexgral_manual_differential_` -
`bool` 184,
466, 467, 474, 504, 512, 532,
549, 579, 612, 641, 647, 667
`\l__intexgral_manual_jacobian_` -
`bool` 185, 469, 470, 483, 496,
521, 541, 558, 572, 588, 604,
621, 635, 656, 675
`\__intexgral_mathchoice:nnnn`
. 204, 775
`\__intexgral_mkern:N` 202,
479, 488, 492, 493, 498, 501,
507, 517, 526, 530, 531, 535,
537, 543, 554, 569, 574, 578,
582, 593, 599, 602, 606, 632,
637, 640, 669, 672, 677
`\__intexgral_new_limits_` -
`group:nn` 980, 993, 998, 1004,
1009
`\__intexgral_new_variables_` -
`group:nnn` . 1011, 1026, 1034,
1041, 1047
`\__intexgral_parse_integral_` -
`keys:n` 386,
1153
`\__intexgral_parse_limits:n` .
. . . . 235, 262, 274, 937
`\__intexgral_parse_variables:`
. 335, 464
`\l__intexgral_primitive_left_` -
`delim_size_tl` 182, 945, 950,
955, 960, 965, 1128, 1131
`\l__intexgral_primitive_left_` -
`delim_tl` . 180, 923, 928, 971,
976, 1129, 1132
`\l__intexgral_primitive_` -
`right_delim_size_tl` . 183,
946, 951, 956, 961, 966, 1134
`\l__intexgral_primitive_` -
`right_delim_tl` 181, 924, 929,
972, 977, 1135
`\l__intexgral_primitive_tl` 179,
1133, 1143
`\__intexgral_print_default_` -
`integral: . . . . .` 472, 701,
705
`\__intexgral_print_default_` -
`integral_inv: . . .` 510, 692,
696
`\__intexgral_print_integral:`
. 682, 1156
`\__intexgral_print_nested_` -
`integral: . . . . .` 547,
702
`\__intexgral_print_nested_` -
`integral_inv: . . . . .` 585,
693
`\__intexgral_print_primitive:`
. 1125, 1144
`\__intexgral_print_product_` -
`integral: . . . . .` 610,
703
`\__intexgral_print_product_` -
`integral_inv: . . . . .` 645,
694
`\__intexgral_retrieve_key_` -
`name:w` 371,
381
`\c__intexgral_right_bracket_` -
`tl` 177, 297, 302, 924,
972
`\l__intexgral_separate_` -
`integral_bool` 187, 446, 713,
718, 723
`\__intexgral_set_limits:nn` . .
. 306, 328
`\__intexgral_set_limits_` -
`regular: . . . . .` 257,
737
`\__intexgral_set_limits_` -
`starred: . . . . .` 269,
744
`\l__intexgral_symbol_inner_` -
`sep_muskip` . . 196, 492, 530,
912
`\l__intexgral_symbol_post_` -
`sep_muskip` 197, 493, 531, 569,

599, 632, 669, 908
`\__integral_trim_pm:w` .. 234, 251, 252
`\l__integral_upper_limit_tl` .. 170, 215, 311, 315, 750, 940, 1137
`\l__integral_upper_limits_seq` .. 224, 265, 293, 323, 330
`\l__integral_variable_sep_muskip` 198, 501, 537, 578, 602, 640, 672, 909
`\l__integral_variables_seq` . 222, 342, 347, 351, 687, 840, 851
`\l__integral_vectorial_differential_bool` 188, 339, 365, 502, 538, 858
`\l__integral_vectorial_differential_style_tl` 172, 366, 913
`\__integral_warning_msg_header:` 83, 118, 125, 132, 139, 155
`\integralsetup` 1114, 1116
`\invertdiff` 1117
iow commands:
`\iow_newline:` 19, 20, 89

J

`\jacobian` 148, 485, 523, 560, 590, 623, 658, 1121

K

keys commands:
`\keys_define:nn` .. 33, 708, 863, 933, 1056, 1072, 1090, 1102
`\keys_if_exist:nnTF` . 160, 388, 1053, 1070, 1088
`\keys_set:nn` 389, 417, 1113, 1115, 1142

L

`\left` 945

M

`\mathbb` 27, 1194
`\mathchoice` 204
`\mathnormal` 56, 75
`\mathrm` 61, 70
msg commands:
`\msg_critical:nn` 25
`\msg_error:nn` . 1120, 1122, 1124

`\msg_error:nnn` . 992, 999, 1025, 1036, 1054, 1082
`\msg_line_context:` 87
`\msg_new:nnn` 16, 115, 123, 130, 137, 144, 147, 150, 153
`\msg_new:nnnn` 91, 95, 99, 103, 107, 111
`\msg_see_documentation_text:n` 21
`\msg_warning:nn` 765, 917
`\msg_warning:nnn` 757, 849, 1052, 1069, 1087

muskip commands:
`\muskip_new:N` 196, 197, 198, 199, 200

N

`\NeedsTeXFormat` 3
`\NewDocumentCommand` 989, 995, 1001, 1006, 1022, 1028, 1038, 1043, 1049, 1066, 1084, 1100, 1112, 1114, 1117, 1139, 1147
`\NewLimitsKeyword` 113, 118, 989, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177
`\NewSymbolKeyword` .. 97, 1049, 1159, 1160, 1161, 1162, 1163, 1164, 1165
`\NewVariableKeyword` 1022, 1178, 1179, 1180, 1181, 1182, 1183
`\NewVarKeyword` 105

O

`\oiint` 1165
`\oint` 1164
`\oint` 1163

P

`\partial` 828
`\phi` 1183
`\pi` 1171, 1172, 1173
prg commands:
`\prg_new_conditional:Npnn` 226
`\prg_return_false:` 232
`\prg_return_true:` 231
`\ProcessKeyOptions` 81
prop commands:
`\prop_get:NnNTF` 836
`\prop_gput:Nnn` .. 163, 982, 1013
`\prop_if_in:NnTF` 875, 896, 991, 1003, 1024, 1040, 1051, 1068, 1086

`\prop_if_in_p:Nn` 239
`\prop_item:Nn` 244, 247, 876, 897
`\prop_new:N` 194, 195
`\prop_pop:NnNTF` 997, 1030
`\prop_remove:Nn` . . . 1008, 1045
`\ProvideLimitsKeyword` 1001
`\ProvidesExplPackage` 11
`\ProvideSymbolKeyword` 1084
`\ProvideVariableKeyword` . . . 1038

Q

quark commands:

`\q_stop` . 205, 234, 245, 251, 252,
 371, 381, 985

R

regex commands:

`\regex_split:nnN` 391
`\RenewLimitsKeyword` 109, 995
`\RenewSymbolKeyword` 93, 1066
`\RenewVariableKeyword` 1028
`\RenewVarKeyword` 101
`\RequirePackage` 4, 27
`\rho` 1182
`\right` 946

S

scan commands:

`\scan_stop:` . 203, 776, 777, 778,
 779

seq commands:

`\seq_count:N` 323, 324, 405, 412,
 436, 566, 596, 629, 664
`\seq_gclear:N` 1033, 1046
`\seq_gpop_left:NNTF` . 562, 616,
 625, 651, 660
`\seq_gset_split:Nnn` 164, 1019
`\seq_if_empty:NTF` . . . 322, 458
`\seq_if_exist:NTF` 843, 879, 900,
 1017
`\seq_item:Nn` 264, 266, 279, 281,
 287, 289, 296, 300, 329, 330,
 375, 393, 416, 419, 420, 423,
 438, 568, 570, 575, 598, 600,
 631, 633, 638, 642, 666, 670,
 673, 678, 939, 941
`\seq_map_indexed_inline:Nn` 271,
 351
`\seq_map_inline:Nn` 259, 789, 810
`\seq_new:N` 190,
 191, 192, 218, 219, 220, 221, 222,
 223, 224, 225, 1018

`\seq_put_right:Nn` 263,
 265, 293, 294, 331, 353, 407,
 413, 454, 460

`\seq_set_eq:NN` 341, 346, 845, 881,
 902

`\seq_set_item:Nnn` 396

`\seq_set_split:Nnn` 165,
 261, 273, 374, 414, 448, 736, 743,
 788, 809, 839, 851, 871, 892, 937

`\seq_use:Nn` 161, 398, 409,
 478, 487, 491, 494, 499, 506,
 516, 525, 529, 534, 540, 544,
 553, 581, 592, 603, 607, 687

`\l_tmpa_seq` . 261, 264, 266, 273,
 279, 281, 287, 289, 296, 300,
 374, 375, 391, 393, 396, 405,
 408, 412, 413, 416, 419, 423, 937,
 939, 941

`\l_tmpb_seq` . 414, 420, 436, 438

`\sin` 1183

str commands:

`\str_case:nnTF` . . 423, 690, 699
`\str_case_e:nnTF` 296, 300
`\str_if_eq:nnTF` . . 158, 159, 392,
 833, 868, 889

`\str_if_eq_p:nn` . 162, 229, 230,
 438

`\str_item:nn` 229, 230

`\str_new:N` 201

`\str_set:Nn` 714, 719, 724

`\str_uppercase:n` 794, 815

T

TeX and L^AT_EX 2_ε commands:

`\@ifpackagelater` 24
`\@onlypreamble` 1116
`\intexgral@date` 8, 13
`\intexgral@description` . . 9, 15
`\intexgral@module` 6, 12
`\intexgral@version` 7, 14

tex commands:

`\tex_left:D` 297, 298
`\tex_limits:D` 45, 730
`\tex_mathop:D` 208
`\tex_mathpunct:D` 280, 288
`\tex_mkern:D` 203, 776, 777, 778,
 779

`\tex_nolimits:D` 47, 732

`\tex_right:D` 301, 302

`\tex_unkern:D` 503, 539

`\theta` 1180, 1181, 1182, 1183

`\thinmuskip` 1188, 1189

`\times` 792, 813
`tl` commands:
 `\c_space_tl` 86
 `\tl_clear:N` 766
 `\tl_const:Nn` 176, 177
 `\tl_head:n` 794, 815
 `\tl_if_blank:nTF` 1015
 `\tl_if_empty:NTF` 791, 812
 `\tl_if_empty:nTF` 1150
 `\tl_if_empty_p:N` 1128
 `\tl_if_eq_p:NN` 1129
 `\tl_if_in:NnTF` 376
 `\tl_if_regex_match:nnTF` . 465, 468
 `\tl_new:N` . . 31, 32, 166, 167, 169, 170, 171, 172, 173, 174, 175, 178, 179, 180, 181, 182, 183
 `\tl_put_right:Nn` 769, 773, 792, 797, 813, 818
 `\tl_set:Nn` 55, 60, 69, 74, 277, 285, 310, 311, 314, 315, 375, 378, 787, 793, 795, 808, 814, 816, 828, 938, 940, 1143, 1155
 `\tl_set_eq:NN` 45, 47, 168, 384, 730, 732, 755, 758, 923, 924, 928, 929, 945, 946, 950, 951, 955, 956, 960, 961, 965, 966, 971, 972, 976, 977
 `\tl_tail:n` 796, 817
 `\tl_trim_spaces:n` . 92, 96, 100, 104, 108, 112, 126
 `\tl_use:N` 299, 563, 617, 626, 652, 661
 `\l_tmpa_tl` 375, 376, 381, 384, 562, 563, 616, 617, 625, 626, 651, 652, 660, 661, 787, 788, 808, 809, 837, 842, 997, 1031
token commands:
 `\c_math_subscript_token` . 212, 822, 1136
 `\c_math_superscript_token` 214, 358, 801, 1137
 `\token_to_str:N` 93, 97, 101, 105, 109, 118, 127

U
use commands:
 `\use_none:nn` 1130

V
`\variables` 151, 686, 1123
`\vec` 1193
`\vert` 929, 977