TENSORSTYLES

This package provides a generic customable display of tensors.

Romain NOEL
romainoel@free.fr

v1.0.1.4 — 2026-04-28

Abstract

They are different notation conventions for tensors. And according to the context (or the local rules) one would like to switch between conventions: keeping the meaning of each tensor, just changing the display. So, this package is here to print tensors according to their order yet independently from their meaning in a very customizable way.

Contents

1 Introduction

tensorstyles was created to solve a problem we regularly faced: switching from one tensor notation to another. Such an action should be trivial but mainstream tensors libraries do not separate content and form, and only offer one or two notations each.

Therefore, if a user wants to switch from one convention to another (let's say when publish in several journals), they will need to manually change all the command related to tensor by hand without changing the content of their documents.

This is why we propose **tensorstyles**, a package that allows the representation of tensors in many notations, and to switch between them seamlessly.

2 Getting started

2.1 Installing from CTAN

The latest stable version of **tensorstyles** is available on CTAN and should now be part of the usual TEX distributions (TEX Live, MacTEX , MikTEX), under the name **tensorstyles**. It means that if your distribution is kept up-to-date, the package should normally be already installed on your system. If this is not the case, consider updating the packages of your Tex distribution.

For TEX Live and MacTEX users, this usually means running

```
tlmgr update --all
```

or if administrative privileges are required

```
sudo tlmgr update --all
```

For MikTEX users, please refer to the official MikTEX documentation.

2.2 Installing from Gitlab

If you want to use the cutting-edge development version of **tensorstyles**, you can install it manually by following these steps:

Download the source from **tensorstyles** repository using `git clone` or as a zip archive of the latest development version.

Compile the style files by running `l3build unpack` inside the downloaded directory. (Or run LATEX directly on `source/tensorstyles.ins`.)

Move the resulting `*.sty` files to the folder containing your presentation. To use **tensorstyles** with many presentations, run `l3build install` or move the `*.sty` files to a folder in your TEX path instead.

Use the package by declaring `\usepackage{tensorstyles}` in the preamble of your document.

tensorstyles uses the l3build system to offer the following installation options for advanced users:

`l3build unpack` builds the theme style files.

`l3build doc` builds this documentation manual and the examples.

`l3build check` builds the theme and manual.

`l3build clean` removes the files generated by `l3build`.

`l3build install` installs the theme into your local texmf folder.

`l3build uninstall` removes the theme from your local texmf folder.

2.3 A minimal example

The following code shows a minimal example **tensorstyles** usage .

```
\documentclass{article}
\usepackage{tensorstyles}
\begin{document}
  $\tensor[preset=einstein]{4}{A}$
\end{document}
```

$$\{A_{a\,b\,c\,d}|$$

Figure 1: Result of a simple example.

2.4 Dependencies

tensorstyles depends on the following standard packages:

- `expl3`
- `amsmath`
- `amssymb`
- `mathtools`

3 Creating and using commands

3.1 Creating a command

The **tensorstyles** package does not only provide a macro to create tensors. The users can create their own.

To do so the package provides a command that acts in a manner similar to a constructor in object-oriented programming; this command is `\DeclaretensorstylesCmd`

It takes two arguments, the first is the name of the macro to create, including the backslash, and the second, optional argument, is a list of options to customize the display.

For example:

```
\DeclareTensorsCmd{\myTensorCmd}[preset=einstein]
\DeclareTensorsCmd{\myOtherTensorCmd}[preset=math, arrows
   =true]
\DeclareTensorsCmd{\thatTensorCmd}[preset=einstein,
   default-indices=contra]
\DeclareTensorsCmd{\anotherTensorCmd}[preset=full]
```

Afterward the newly created command can be used.

3.2 Using a custom command

Once a command is created it can be used anywhere in the document afterward. All the commands take the same arguments.

`\tensor[<option>]{<rank>}{<symbol>}[<indices>]_{sub}^{sup}`

or `\myTensorCmd[<option>]{<rank>}{<symbol>}[<indices>]_{sub}^{sup}`

`#1` : `options`: an optional list of options to override for this call
`#2` : `rank`: the rank of the tensor
`#3` : `symbol`: the symbol that represents the tensor
`#4` : `indices`: an optional list of Einstein indices
`#5` : `sub` and `sup`: optional sub- and super-scripts to add after the tensor

The options argument accepts any and all of the options that can be defined during command creation. The options given will override the ones that were defined when the command was created, but only for this call.

The rank can be a number or a letter. It is used for determining the font to use, which arrow type to add, and how many indices to write. For fonts and arrows, this is determined by a list, customizable via an option, that associates a given rank with a font and an arrow. For indices, unless they were manually given, the macro will add as many indices as the rank specified if it is a number, or if it is a letter the rank itself will be added instead. There is one exception to this: x and X. When used, the macro will calculate the rank based on the number of indices given, potentially 0.

The symbol can be anything, including another tensor; it is the object to which the notations will be added.

Indices are given with a specific syntax:

- First, write either ^ for contravariant indices or _ for covariant indices.
- Then any character that follows will be added as an index; for instance ^abc adds three contravariant indices a, b, and c.
- Any subsequent ^ and _ is used to switch between covariant and contravariant indices.
- If a group, {...}, is given it is treated as one index and it is expanded, allowing indices to be given with sub- and superscripts, or even passing a tensor as an index.

Every created command also has a star variant. This variant has another, fully customizable, set of options. This allows one to rapidly switch between two conventions.

4 Options

All the options have a star version used for the star variant of the command. They take the same arguments and have the same effects. Their name is the name of the regular options with a star added at the end. For example, `preset` and `preset*`.

The default values of each option depend on the preset.

4.1 Local vs. global options

The options given directly to the command (as described in the section 3.2) are applied locally. So they will only affect the display of this particular command, not the following ones.

For global setting that will last until the next modification, one can use

`\tensorsseet{<command>}[<global␣options>]`

`#1` : `command` for which the options have to be set.
`#2` : `global options` that will be set.

This setting can be called at any moment within the document.

4.2 General options

`preset` *empty, full, math, arrow, einstein, engineer, bra, ket* ..... full

Which preset to use.
`\tensor[preset=empty]{2}{x}` gives x
`\tensor[preset=full]{2}{x}` gives $\left\{ \overleftrightarrow{X}_{\alpha\,\beta} \right|$
`\tensor[preset=math]{2}{x}` gives X
`\tensor[preset=arrow]{2}{x}` gives $\overleftrightarrow{x}$
`\tensor[preset=einstein]{2}{x}` gives $\{x_{ab}|$
`\tensor[preset=engineer]{2}{x}` gives $\underline{\underline{x}}$
`\tensor[preset=bra]{2}{x}` gives $\langle x|$
`\tensor[preset=ket]{2}{x}` gives $|x\rangle$

`switch-*` *true, false* ..... false

Option to invert the star switch and regular version. `\tensor*{2}{x}` is equivalent to `\tensor[switch-*=true]{2}{x}`.
Every option has a star equivalent that controls the display of the star-versioned command. Therefore, one can use a combination of `preset` and `preset*` to switch easily between two displays.
For example, with the global setup: `\tensorsset{\tensor}[preset=math, preset*=einstein]`, the commands `$\tensor{2}{x}$`, `$\tensor[switch-*=true]{2}{x}$` and `$\tensor*{2}{x}$` yields : X, x_{ab} and x_{ab}

4.3 Font options

`font` *true, false* ..... true

Whether a special font should be applied to tensors.

`font-cmd` *{0=\font, 1=\font, ...}* .....

A list containing pairs `rank=font`. It contains the fonts that will be used for a tensor of a given rank. If the user tries to create a tensor with a rank not defined here and font is activated, the package will throw an error.
For example, `font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\MakeBoldUppercase, n=\mathfrak}` is used for `preset=math`

4.4 Arrows options

`arrows` *true, false* ..... true

Whether arrows should be added to tensors.

`arrow-cmd` *{0=\macro, 1=\macro, ...}* .....

A list containing pairs `rank=macro`. It contains the macro that will be used to add arrows for a tensor of a given rank. If the user tries to create a tensor with a rank not defined here and arrows are activated, the package will throw an error.
For example, `arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftrightarrow, 3=\overrightleftrightarrow, 4=\overdoubleleftrightarrow, 5=\overrightdoubleleftrightarrow, n=\underleftrightarrow}` is used for `preset=arrow`

`recursive-arrows` *true, false* ..... false

Whether to add as many arrows as the rank.

`recursive-arrows-cmd` *\macro* .....

The macro that will be used to add arrows recursively, for example `recursive-arrows-cmd=\underline` is use for `preset=engineer`.

4.5 Einstein options

`contra-variant` *true, false* ..... true

Whether indices should be added to tensors.

`default-indices` *contra, covariant* ..... covariant

Whether indices added automatically should be covariant or contravariant.
For example: `\tensor[default-indices=contra]{2}{x}` gives $\left\{\overleftrightarrow{X}^{ab}\right|$

`contra-style` *greek, alph, arabic* ..... alph

The alphabet to use when adding contravariant indices automatically. Either Greek or Latin alphabets, or numbers.
For example: `\tensor[contra-style=arabic,default-indices=contra]{2}{x}` gives $\left\{\overleftrightarrow{X}^{12}\right|$

`covariant-style` *greek, alph, arabic* ..... greek

The alphabet to use when adding covariant indices automatically. Either Greek or Latin alphabets, or numbers.
For example: `\tensor[covariant-style=arabic]{2}{x}` gives $\left\{\overleftrightarrow{X}_{12}\right|$

`specific-indices` *1,2,3,...* .....

When adding indices automatically, which ones should not be added as `default-indices`. For instance, if default-indices is contra, which indices to add as covariant indices.
For example: `\tensor[specific-indices=2]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}_{\alpha}{}^{b}{}_{\gamma}\right|$

`collapsed-indices` *true, false* ..... false

Whether indices are spaced horizontally, *i.e.* there will not be a covariant index under a contravariant index and vice versa, or not.
For example: `\tensor[specific-indices=2,collapsed-indices=true]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}^{b}_{\alpha\,\gamma}\right|$

`index-marker` *\␣, ., ;, ...* ..... \␣

If indices are not collapsed, what symbol to use to fill in the blanks.
For example: `\tensor[specific-indices=2,index-marker=.]{3}{x}` gives $\left\{\widetilde{\mathcal{X}}_{\alpha\,.\,\gamma}^{\,b\,.}\right|$

4.6 Dirac options

`bra-ket` *true, false* ..... true

Whether delimiters should be added to tensors.

`scale-var` *auto, none, big, bigg, Big, Bigg* ..... auto

How the delimiters should scale with the tensor.

`delims-var` *\rvert\lbrace, \rbrace\lvert, ...* ..... \lbrace\rvert

Which delimiters to add. If scale-var is auto, the delimiters must be usable with \left and \right, else any symbol can be used.

`mix-delims` *true, false* ..... true

If two tensors are following each other, and the delimiters between them are identical, whether to display one or two.
For example: `\tensor[preset=bra]{1}{x}\tensor[preset=ket]{1}{y}` gives $\langle x|y\rangle$ while `\tensor[preset=bra, mix-delims=false]{1}{x}\tensor[preset=ket]{1}{y}` gives $\langle x||y\rangle$

5 Using the package

In this section, we will take the example of a user trying to write an article in which they will need several tensor notations.

Let's say that the user wants to represent tensors using an Einstein convention. More specifically they want to represent purely covariant and purely contravariant tensors with this convention.

There are two ways to do that with **tensorstyles**. The first is to define two new commands to represent tensors, one for covariant tensors and one for contravariant tensors.

```
\DeclareTensorsCmd{\covariantTensor}[preset=einstein]
\DeclareTensorsCmd{\contravariantTensor}[preset=einstein,
    default-indices=contra]
```

The second is to take advantage of the star variant of the commands and to define only one that does both.

```
\DeclareTensorsCmd{\myTensor}[preset=einstein, preset*=
   einstein, default-indices*=contra]
```

We will take the second approach for this example.

Now whenever they want to represent a tensor in their article, all they have to do is call this new command:

`\myTensor{4}{e}` $\rightarrow e_{a\,b\,c\,d}$ `\myTensor*{4}{e}` $\rightarrow e^{a\,b\,c\,d}$

Unfortunately, the user now needs to represent a tensor with both covariant and contravariant indices, something they did not anticipate. Fortunately **tensorstyles** offers two ways to do just that. First, the user could use an option to specify which indices should be contravariant or covariant:

`\myTensor{4}{e}[specific-indices={1,4}]` $\rightarrow e^{a}{}_{b\,c}{}^{d}$ Which is equivalent to
`\myTensor*{4}{e}[specific-indices*={2,3}]` $\rightarrow e^{a}{}_{b\,c}{}^{d}$

Or they can use an optional argument of the command to pass the indices directly.

`\myTensor{4}{e}[^a_bc^d]` $\rightarrow e^{a}{}_{bc}{}^{d}$ Which is equivalent to
`\myTensor*{4}{e}[^a_bc^d]` $\rightarrow e^{a}{}_{bc}{}^{d}$

This is more powerful since the user will have full control over the positions of the indices and their values. This is especially useful to pass indices with sub- or superscripts or even an entire command as an index. For example

`\myTensor{4}{e}[^{a_i}_bc^{\myTensor{2}{a}}]` $\rightarrow e^{a_i}{}_{bc}{}^{a_{a\,b}}$

Our user is now quite far into their document, but now they want to represent a tensor with another convention, let's say an engineer notation with recursive underlines. Perhaps they want to present their document to students, and to show them that other notations exist.

They can do this in two ways. First they can define a new command

```
\DeclareTensorsCmd{\tensorEngineer}[preset=engineer]
```

This works quite well and it preserves the separation of content and form. But it adds a new command that will only be used once, and so it is not really optimal in terms of memory or computation.

Alternatively they can change the behavior of their already existing command locally:

`\myTensor{4}{e}[preset=engineer]` $\rightarrow \underline{\underline{\underline{e}}}$

This avoids the declaration of a new command but it partially links content and form, so our user may have to be careful if they want to change the notation of their document.

For the last part of their document our user wants to use another convention entirely. Moreover it is a complex one: they want to use delimiters, like a Dirac convention, and to change the font of the tensor depending on the rank. Since they want this behavior for the rest of their document, redefining the display at each call is out of the question. So they have two options:

They can define a new command for this convention, or, the approach we will follow, they can use the `\tensorsset` command to override the options globally after its call.

```
\myTensor{2}{a}

\tensorsset{\myTensor}[preset=math, bra-ket=true, delims-
    var = \lbrace\rvert]

\myTensor{4}{e} and \myTensor{2}{a}
```

Gives us:

a_{ab}

$\{\mathbb{E}|$ and $\{A|$

Now, let us say that users want to switch the notation convention within the whole document. Thanks to **tensorstyles** this is trivial, all they need to do is go back to the function declaration and change the options at the beginning of the document they gave:

```
\DeclareTensorsCmd{\myTensor}[preset=arrow, preset*=math]
```

Plus they also used an engineer preset once, which will not be changed either, but since this call was to explicitly show this notation there is no need for a change. However, if many different presets have been used through the document, the previous approach remains tedious. Therefore, another approach can be to redefine presets themselves.

6 Sources of inspiration

Many packages have been used as sources of inspiration to build **tensorstyles**:

- https://gitlab.com/RomainNOEL/latex3_template_pkg for LateX3 template.
- mattens package from Danie Els for complex vector notations.
- hhtensor package from Harald Harders to print tensors with arrow, bold, or uline.
- tensor package from Philip G. Ratcliffe for Einstein tensor notation.
- tensind package from Javier Bezos for Einstein tensor notation.
- overarrows package from Julien Labbé to draw arrows that stretch.
- braket package from Donald Arseneau for Dirac bra-ket notation.
- physics package from Sergio C. de la Barrera for vec, braket and too many things.

7 Known issues

Some limitations have been faced.

- when using `\dots` with `collapsed-indices`, this is due to the tentative of context determination employed by `\dots` which interferes with the **tensorstyles**. A simple workaround is to use `\ldots` instead.
- if you are mixing delimiters using two variables of different sizes with automatic scale, *e.g.* `\tensor[preset=bra, mix-delims=true, scale-var=auto]{n}{\frac{1}{A}}\tensor*[preset*=ket, mix-delims*=true, scale-var*=auto]{n}{B}`, then the delimiters will not have the same size. A simple solution is to set the desired scale for the "wrong" delimiter.

7.1 Contributors

- Maelwen THOMAS
- Laurent NAVARRO

8 License

tensorstyles is licensed under the terms of the LaTeX project public license (LPPL) 1.3c license.

9 Commands description for users

PACKAGE

\usepackage[options]{tensorstyles}

\usepackage[⟨option1=value1, ...⟩]{⟨tensorstyles⟩}
where the options are (default marked as default):
⟨footer template⟩ = [⟨tensorstyles | ... ⟩]

Description of usepackage{tensorstyles} which loads the package.

(*End of definition for* \usepackage[options]{tensorstyles}. *This function is documented on page* ??.)

Key-val: tensorstyles

\tensorsset{all}[options]

\nameMacro[⟨option1=value1, ...⟩]{⟨tensor-cmd⟩}
where the options are (default marked as default):
⟨scale-auto⟩ = [⟨leftright | auto ⟩]

The option scale-auto that can be set for all command groups, can take leftright or mleftmright values. This option is controlling the delimiters to use in auto mode (\left\right or \mleft \mright from mleftright package).

(*End of definition for* \tensorsset{all}[options]. *This function is documented on page* ??.)

Setting keys

'TENSOR' GROUP COMMAND CODE

Command Declaration

\DeclareTensorsCmd

New: 2025-02-17

#1 : command name to be declared.
#2 : keyval (option) to be given to the new command as a first (non-default) setup.

Command to declare a new command from a command group, with eventually options to be given to this new option. Such new definition are useful for users who would like to create their own custom command that will have the same behaviour than the ones furnished by the package.

Rest of the Code (common)

Scale, delimiters, evaluation

\MakeBoldUppercase	\MakeBoldUppercase, MakeUpperBB, MakeUpperCal, MakeUpperFrak, overdoubleleftrightarrow, overr
\MakeUpperBB	
\MakeUpperCal	
\MakeUpperFrak	
\overdoubleleftrightarrow	
\overrightleftrightarrow	
\overrightdoubleleftrightarrow	

Commands applying a combination of font commands or glyph (arrow) commands. The combinations are quite explicit from the command names.

Generate variants

Messages

Declaring & Generate new group of commands

10 Package implementation for developers

```
⟨*package⟩
%\ProvidesExplPackage{tensorstyles}{2026-04-28}{1.0.1.4}{This package provides a generic cus
\RequirePackage{expl3}
% \RequirePackage{xparse,l3keys2e}% depreciated now.
\RequirePackage{amsmath}% \overrightarrow
\RequirePackage{amssymb}% \curvearrowright
% \RequirePackage{bm}% recommended for bold math
\RequirePackage{mathtools}% for \mathmakebox
```

PACKAGE

\c__tensorstyles_digits_tl Constant variables containing the digits.

```
\tl_const:Nn \c__tensorstyles_digits_tl {123456789-0}
```

(*End of definition for* `\c__tensorstyles_digits_tl`.)

\usepackage[options]{tensorstyles} `#1 : no package option` are present for now.

Description

The following test files are used for this code: tensorstyles-test-003.

```
% Define keys/options for the package.
\keys_define:nn { tensorstyles / pkg }
{
 % italic .choice:,
 % italic .usage:n = { load },
 % italic .default:n = { true },
 % italic / true .code:n =
 % {
 %  \bool_set_true:N  \l__tensorstyles_pkg_italic_bool
 %  \bool_set_false:N \l__tensorstyles_pkg_upright_bool
 % },
 % italic / false .code:n =
 % {
 %  \bool_set_false:N \l__tensorstyles_pkg_italic_bool
 %  \bool_set_true:N  \l__tensorstyles_pkg_upright_bool
 % },
 % italic / unknown .code:n = { \msg_error:nnx { tensorstyles } { boolean-values-only } \l_k
 % %
 % upright .choice:,
 % upright .usage:n = { load },
 % upright .default:n = { true },
 % upright / true .code:n =
```

```
% {
%  \bool_set_true:N  \l__tensorstyles_pkg_upright_bool
%  \bool_set_false:N \l__tensorstyles_pkg_italic_bool
% },
% upright / false .code:n =
% {
%  \bool_set_false:N \l__tensorstyles_pkg_upright_bool
%  \bool_set_true:N  \l__tensorstyles_pkg_italic_bool
% },
% upright / unknown .code:n = { \msg_error:nnx { tensorstyles } { boolean-values-only } \l_
% upright .initial:n = true,
}
% process the options.
\ProcessKeyOptions[ tensorstyles / pkg ]
```

(*End of definition for* `\usepackage[options]{tensorstyles}`*. This function is documented on page* **??**.)

Key-val: tensorstyles

`\l__tensorstyles_all_all_user_keys_prop` `\c__tensorstyles_all_pkg_keys_prop`

Property list containing the default value for options that are share by all command groups; and a list that will receive the values given by users for these options.

The following test files are used for this code: tensorstyles-test-003.

```
\prop_const_from_keyval:Nn \c__tensorstyles_all_pkg_keys_prop {scale-auto = leftright}
\prop_new:N \l__tensorstyles_all_all_user_keys_prop
```

(*End of definition for* `\l__tensorstyles_all_all_user_keys_prop` *and* `\c__tensorstyles_all_pkg_keys_prop`.)

`\tensorsset{all}[options]`

`#1` : `width` Name of the option to add, this name should also correspond to the name of the environment followed by the suffix 'env'.

Option available for all command groups controlling the delimiters in automatic scale mode.

The following test files are used for this code: tensorstyles-test-005.

```
% Define the options for all group commands (package included).
\keys_define:nn { tensorstyles / all / all } {
 scale-auto .choice:,
 scale-auto / leftright .code:n = {
  \cs_set_eq:NN \__tensorstyles_auto_left:n \left
  \cs_set_eq:NN \__tensorstyles_auto_right:n \right
 },
 scale-auto / mleftmright .code:n = {
```

```
  \cs_set_nopar:Npn \__tensorstyles_auto_left:n  { \mleft  }
  \cs_set_nopar:Npn \__tensorstyles_auto_right:n { \mright }
 }
}
```

(*End of definition for* `\tensorsset{all}[options]`. *This function is documented on page* **??**.)

`\l__tensorstyles_new_var_seq` Local sequence storing new variables names to create.

```
\seq_new:N \l__tensorstyles_new_var_seq
```

(*End of definition for* `\l__tensorstyles_new_var_seq`.)

`\__tensorstyles_new`

New: 2025-02-17

`#1` : `variable-type` eg. `tl` (token list), `dim`, `seq`, etc.
`#2` : `grp-cmd+variant` eg. `$tensor_tensor$`
`#3` : `variable-category` eg. `style`
`#4` : `cs-var-list` comma separated list of variable names to create.

This function is creating the local variables `\l__tensorstyles_'type'_'variant'_'cat'_'var'` for all the variable names given in the argument 4 (`cs-var-list`).

The following test files are used for this code: tensorstyles-test-005.

```
\cs_new:Npn \__tensorstyles_new:nnnn #1 #2 #3 #4 {
 \seq_set_from_clist:Nn \l__tensorstyles_new_var_seq { #4 }
 \seq_map_inline:Nn \l__tensorstyles_new_var_seq
 { \cs:w #1_new:c \cs_end: { l__tensorstyles_#2_#3_##1_#1 } }
}
```

Setting keys

__tensorstyles_set_default_from_keyval

New: 2025-02-17

#1 : `variable-type` eg. tl (token list), dim, seq etc.

#2 : `preset` eg. preset

#3 : `keyval` the option given

#4 : `cs-var-list` comma separated list of variable names to create.

This function is setting the default keys/options from a list of keyval.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_set_default_from_keyval: variant, preset, keyval,  group command
\cs_new_protected:Npn \__tensorstyles_set_default_from_keyval:nnnn #1 #2 #3 #4 {
 % tensorstyles~set~default~from~keyval
 % Check if keyvalues have been given.
 \tl_if_novalue:nTF { #3 }
 { % no user keys given so _user_keys_prop is taking pkg prop
  \prop_set_eq:cc { l__tensorstyles_#4_#1_user_keys_prop } { c__tensorstyles_#4_#2_pkg_keys_
 }{ % user key given and put it in prop
  \prop_put_from_keyval:cn { l__tensorstyles_#4_#1_user_keys_prop } { #3 }
 }% end if
 %
 % prepare the map of keys
 \__tensorstyles_set_default_auxi:cnn { l__tensorstyles_#4_#1_user_keys_prop } { #1 } { #4 }
 %
 % set this map as default
 \keys_set:nn { tensorstyles/#4/#1 } { default }
}
\cs_generate_variant:Nn \__tensorstyles_set_default_from_keyval:nnnn { nVnn }
```

__tensorstyles_set_default_auxi
__tensorstyles_set_default_auxii

New: 2025-02-17

#1 : `prop-key-val` eg. tl (token list), dim, seq etc.
#2 : `variant` eg. 'options=toto'
#3 : `grp-cmd` eg. `style`

These functions are internally used to set the default keys/options.

The following test files are used for this code: tensorstyles-test-005.

```
\cs_new_protected:Npn \__tensorstyles_set_default_auxi:Nnn #1 #2 #3 {
 \__tensorstyles_set_default_auxii:fnn { \prop_to_keyval:N #1 } { #2 } { #3 }
}
\cs_generate_variant:Nn \__tensorstyles_set_default_auxi:Nnn { c }
%
% key-val, variant,  group command
\cs_new_protected:Npn \__tensorstyles_set_default_auxii:nnn #1 #2 #3 {
 \keys_define:nn { tensorstyles/#3/#2 } { default .meta:n = { #1 } }
}
\cs_generate_variant:Nn \__tensorstyles_set_default_auxii:nnn { f }
```

\tensorsset

New: 2025-02-17

#1 : `command` for which options will be set or reset.

#2 : `options` input given as setting options.

Function to (re)set keys/options by default for a group of commands.

The following test files are used for this code: tensorstyles-test-005.

```
% \tensorsset m:command:#1 o:options:#2
\DeclareDocumentCommand{\tensorsset}{ m o } {
 \str_if_eq:nnTF { #1 } { all }
 {
  % True: if the 'all' commands have to be set.
  \__tensorstyles_set_default_from_keyval:nnnn { all } {empty} { #2 } { all }
  \keys_set:nn { tensorstyles/all/all } { default }
 }{
  %False: if the command name is given then apply to it:
  % capture the command name.
  \tl_set:Nx \l__tensorstyles_tensorsset_tl { \cs_to_str:N #1 }
  % apply option to the command.
  \tensorstyles_set_keys:Vn \l__tensorstyles_tensorsset_tl { #2 }
 }
}
```

\tensorstyles_local_keys

New: 2025-02-17

#1 : `keyval` for which options will be set or reset.

#2 : `grp-cmd` input given as setting options.

#3 : `command` input given as setting options.

#4 : `preset` of options.

Set options according to the keys-value given by the user locally. This command is used in the code definition of the 'command'-group, ie., is called every time the 'command' is called.

The following test files are used for this code: tensorstyles-test-005.

```
% tensorstyles_local_keys:nnnn #1=keyval, #2=group-command, #3=command, #4=preset
\cs_new_protected:Npn \tensorstyles_local_keys:nnnn #1 #2 #3 #4 {
 % in tensorstyles~local~keys
 \tl_set:Ne \l_tmpa_tl { #4 }

 \tl_if_novalue:nTF { #1 }
 {
  % keyval NOT given
  \bool_if:cT { l__tensorstyles_#2_#3_local_keys_bool }
  {
   % True
   \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
   \bool_set_false:c { l__tensorstyles_#2_#3_local_keys_bool }
  }
 }{
  % keyvalue given
  \bool_if:cTF { l__tensorstyles_#2_#3_local_keys_bool }
  {
   % True
   \tl_put_right:Nn \l_tmpa_tl { , #1 }  % append #1 unexpanded
   \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
  }{
   % False
   \tl_put_right:Nn \l_tmpa_tl { , #1 }  % append #1 unexpanded
   \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
  }
  \bool_set_true:c { l__tensorstyles_#2_#3_local_keys_bool }
 }
}
\cs_generate_variant:Nn \keys_set:nn { nV }
```

\tensorstyles_local_preset

New: 2025-02-17

#1 : `keyval` for which options will be set or reset.

#2 : `grp-cmd` input given as setting options.

#3 : `preset` of options.

#4 : `variant` of options.

Set options according to the keys-value given by the user locally. This command is used in the code definition of the 'command'-group, ie., is called every time the 'command' is called.

The following test files are used for this code: tensorstyles-test-005.

```
% tensorstyles_local_preset:nnn #1=group-command, #2=preset, #3=star-preset #4=variant
\cs_new_protected:Npn \tensorstyles_local_preset:nVVnn #1 #2 #3 #4 {
 \tl_if_eq:NnTF { #2 } { None } {
  \keys_define:nn { tensorstyles/tensor/#1 } { pres .meta:n = { default }}
 } {
  \prop_set_eq:cc { l__tensorstyles_#4_#1_user_keys_prop } { c__tensorstyles_#4_#2_pkg_keys_
  \keys_define:nn { tensorstyles/tensor/#1 } {
   pres .code:n = {
    \prop_map_inline:cn { l__tensorstyles_#4_#1_user_keys_prop }
     { \keys_set:nn { tensorstyles/tensor/#1 } { ####1 = {####2} } }
   }
  }
 }
 \tl_if_eq:NnTF { #3 } { None } {
  \keys_define:nn { tensorstyles/tensor/#1 } { pres* .meta:n = { default }}
 } {
  \prop_set_eq:cc { l__tensorstyles_#4_#1*_user_keys_prop } { c__tensorstyles_#4_#3_pkg_keys
  \keys_define:nn { tensorstyles/tensor/#1 } {
   pres* .code:n = {
    \prop_map_inline:cn { l__tensorstyles_#4_#1*_user_keys_prop }
     { \keys_set:nn { tensorstyles/tensor/#1 } { ####1* = {####2} } }
   }
  }
 }
}
```

\tensorstyles_addto_globalvar_and_setkeys:NNnnn

New: 2025-02-17

#1 : `tl-tensor` token list used to concatenate the name of the command.
#2 : `macro` input given as setting options.
#3 : `command` input given as setting options.
#4 : `keyval` for which options will be set or reset.
#5 : `grp-cmd` input given as setting options.

Set options according to the keys-value given by the user locally. This command is used in the code definition of the 'command'-group, ie., is called every time the 'command' is called.

The following test files are used for this code: tensorstyles-test-005.

```
%
% \tensorstyles_addto_globalvar_and_setkeys:NNnnn #1=tl-tensor, #2=macro, #3=inf, #4=keyval,
\cs_new_protected:Npn \tensorstyles_addto_globalvar_and_setkeys:NNnnn #1 #2 #3 #4 #5 {
 % store
 \tl_set:Nx #1 { \cs_to_str:N #2 }

 % Check if the keyval has values.
 \tl_if_novalue:nTF { #4 }
 {
  % if empty then create the variables by default.
  \__tensorstyles_preamble_aux:Vnnnn #1 { empty } { empty } { } { #5 }
 }{
  % otherwise, create then set the variables.
  \use:c { __tensorstyles_#5_define_preset_keys:n } { #1 }
  \keys_set:nn {tensorstyles/preset/#1} { #4 }
  \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_preset  } }
  \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_preset } }

  \__tensorstyles_preamble_aux:VVVnn #1 \l_tmpa_tl \l_tmpb_tl { #4 } { #5 }
  % \__tensorstyles_preamble_aux:Veenn #1 { full } { empty } { #4 } { #5 }
 }

}
%
% \__tensorstyles_preamble_aux:nnnn #1=variant, #2=preset, #3=star-preset, #4=key-value, #5=
\cs_new_protected:Npn \__tensorstyles_preamble_aux:nnnnn #1 #2 #3 #4 #5 {
 % Check if the command is in the global sequence
 \seq_if_in:cnF { l__tensorstyles_#5_variant_seq } { #1 }
 {

  % if not present
  % add the command to the sequence
  \seq_put_left:cn { l__tensorstyles_#5_variant_seq } { #1 }
  % create new variables (tl) for the new command
  \use:c { __tensorstyles_#5_variables:n   } { #1 }
  \use:c { __tensorstyles_#5_variables:n   } { #1* }
  % define keys for this command
  \use:c { __tensorstyles_#5_define_keys:n } { #1 }
```

'TENSOR' GROUP COMMAND CODE

\l__tensorstyles_tensor_variant_seq
\l__tensorstyles_tensor_rank_holder_tl
\l__tensorstyles_tensor_rank_counter_int

`\l__tensorstyles_tensor_variant_seq` is a sequence to store all the command of this type. `\l__tensorstyles_tensor_rank_holder_tl` is storing the value of the tensor rank given. `\l__tensorstyles_tensor_rank_counter_int` is storing the local value of the rank counter (as Einstein indices are running for example).

```
\seq_new:N \l__tensorstyles_tensor_variant_seq
\tl_new:N \l__tensorstyles_tensor_rank_holder_tl
\int_new:N \l__tensorstyles_tensor_rank_counter_int
```

(*End of definition for* `\l__tensorstyles_tensor_variant_seq`, `\l__tensorstyles_tensor_rank_holder_tl`, *and* `\l__tensorstyles_tensor_rank_counter_int`.)

__tensorstyles_tensor_variables:n

New: 2025-02-17

#1 : variant for which variables have to be declared.

Define variables of the 'tensor' command-group, according to the keys-value given by the user locally.

The following test files are used for this code: tensorstyles-test-005.

```
% Define variables for the tensor group.
% \__tensorstyles_tensor_variables:n variant
\cs_new_protected:Npn \__tensorstyles_tensor_variables:n #1 {
 %%\__tensorstyles_new:nnnn data-type, grp-cmd+variant, category, cs-var-list
 %
 % switch
 \__tensorstyles_new:nnnn { bool } {tensor_#1} { switch } { star }
 \__tensorstyles_new:nnnn { bool } {tensor_#1} { switch } { font, einstein, dirac, arrows, r
 % font
 \__tensorstyles_new:nnnn { prop } {tensor_#1} { font } { cmd }
 % Einstein
 \__tensorstyles_new:nnnn { tl } {tensor_#1} { indexstyle } { covariant, contra }
 % arrows
 \__tensorstyles_new:nnnn { prop } {tensor_#1} { arrow } { cmd }
 % recursive arrows
 \__tensorstyles_new:nnnn { tl } {tensor_#1} { recursive } { arrowCmd }
 % Dirac bra-ket
 \__tensorstyles_new:nnnn { tl } {tensor_#1} { scale  } { var }
 \__tensorstyles_new:nnnn { tl } {tensor_#1} { delims } { eval }
 % keys
 \__tensorstyles_new:nnnn { prop } {tensor_#1} { user  } { keys }
 \__tensorstyles_new:nnnn { bool } {tensor_#1} { local } { keys }
}
```

__tensorstyles_tensor_define_keys:n

#1 : variant for which key corresponding the command associated, have to be declared.

Set options for the 'tensor' group-command according to the keys-value given by the user locally.

The following test files are used for this code: tensorstyles-test-005.

```
% Define keys/options for tensor group command style.
% \__tensorstyles_tensor_define_keys:n variant
\cs_new:Npn \__tensorstyles_tensor_define_keys:n #1 {
 \keys_define:nn { tensorstyles/tensor/#1 } {
```

```
switch-* .bool_set:c = { l__tensorstyles_tensor_#1_switch_star_bool },
preset .code:n = {},
deduce-var .tl_set:c = { l__tensorstyles_tensor_#1_deduce_var_tl },

% font
font .bool_set:c = { l__tensorstyles_tensor_#1_switch_font_bool },
font-cmd .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1_font_cmd_prop } {

% arrow
arrows .bool_set:c = { l__tensorstyles_tensor_#1_switch_arrows_bool },
arrow-cmd .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1_arrow_cmd_prop }

recursive-arrows .bool_set:c = { l__tensorstyles_tensor_#1_switch_recursiveArrows_bool },
recurs-arrows-cmd .tl_set:c  = { l__tensorstyles_tensor_#1_recursive_arrowCmd_tl },

% bra-ket
bra-ket .bool_set:c = { l__tensorstyles_tensor_#1_switch_dirac_bool },
scale-var .choices:nn = { auto, none, big, Big, bigg, Bigg } { \__tensorstyles_set_scale:n
scale-var .default:n = { auto },
scale-var .initial:n = { auto },
delims-var .tl_set:c  = { l__tensorstyles_tensor_#1_delims_var_tl },
mix-delims .bool_set:c = { l__tensorstyles_tensor_#1_mix_delimiter_bool },

% Einstein
contra-variant .bool_set:c = { l__tensorstyles_tensor_#1_switch_einstein_bool },
% \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= covariant or contra, #3= op
covariant-style .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nn
covariant-style .default:n = { greek },
covariant-style .initial:n = { greek },
contra-style .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn {
contra-style .default:n = { alph },
contra-style .initial:n = { alph },
default-indices .choices:nn = { covariant, contra } { \__tensorstyles_set_index_type:nn {#
default-indices .default:n = { covariant },
default-indices .initial:n = { covariant },
index-marker .tl_set:c = { l__tensorstyles_tensor_#1_index_marker_tl },
specific-indices .clist_set:c = { l__tensorstyles_tensor_#1_specific_indices_clist },
collapsed-indices .bool_set:c = { l__tensorstyles_tensor_#1_collapsed_indices },
index-offset .int_set:c = { l__tensorstyles_tensor_#1_index_offset },

%%% Star Version
%%%%%%%%%%%%%%%
```

```
  % switch-** .bool_set:c = { l__tensorstyles_tensor_#1*_switch_star_bool },
  switch-** .code:n = { printSwitchStarStar },
  preset* .code:n   = {},
  deduce-var* .tl_set:c = { l__tensorstyles_tensor_#1*_deduce_var_tl },

  % font
  font* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_font_bool },
  font-cmd* .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1*_font_cmd_prop }

  % arrow
  arrows* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_arrows_bool },
  arrow-cmd* .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1*_arrow_cmd_prop

  recursive-arrows* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_recursiveArrows_bool }
  recurs-arrows-cmd* .tl_set:c  = { l__tensorstyles_tensor_#1*_recursive_arrowCmd_tl },

  % bra-ket
  bra-ket* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_dirac_bool },
  scale-var* .choices:nn = { auto, none, big, Big, bigg, Bigg } { \__tensorstyles_set_scale:
  delims-var* .tl_set:c  = { l__tensorstyles_tensor_#1*_delims_var_tl },
  mix-delims* .bool_set:c = { l__tensorstyles_tensor_#1*_mix_delimiter_bool },

  % Einstein
  contra-variant* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_einstein_bool },
  % \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= covariant or contra, #3= op
  covariant-style* .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:n
  contra-style* .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn
  default-indices* .choices:nn = { covariant, contra } { \__tensorstyles_set_index_type:nn {
  index-marker* .tl_set:c = { l__tensorstyles_tensor_#1*_index_marker_tl },
  specific-indices* .clist_set:c = { l__tensorstyles_tensor_#1*_specific_indices_clist },
  collapsed-indices* .bool_set:c = { l__tensorstyles_tensor_#1*_collapsed_indices },
  index-offset* .int_set:c = { l__tensorstyles_tensor_#1*_index_offset },
 }
}
```

(*End of definition for* `\__tensorstyles_tensor_define_keys:n`.)

__tensorstyles_tensor_define_preset_keys:n
__tensorstyles_tensor_define_local_preset_keys:n

New: 2025-02-12

Updated: 2025-02-12

`#1` : `variant` of the ‘tensor‘ group-command.

Define the keys and variables for presets.

The following test files are used for this code: tensorstyles-test-005.

```
\cs_new:Npn \__tensorstyles_tensor_define_preset_keys:n #1 {
 % Initialize variables to "empty" as baseline
 \tl_set:cn { l__tensorstyles_tensor_#1_preset  } { empty }
 \tl_set:cn { l__tensorstyles_tensor_#1*_preset } { empty }
 \keys_define:nn { tensorstyles/preset/#1 } {
  preset  .tl_set:c  = { l__tensorstyles_tensor_#1_preset  } ,
  preset* .tl_set:c  = { l__tensorstyles_tensor_#1*_preset } ,
  unknown .code:n    = {}
 }
}

\cs_new:Npn \__tensorstyles_tensor_define_local_preset_keys:n #1 {
 % Initialize variables to "empty" as baseline
 \tl_set:cn { l__tensorstyles_tensor_#1_local_preset  } { None }
 \tl_set:cn { l__tensorstyles_tensor_#1*_local_preset } { None }
 \keys_define:nn { tensorstyles/preset/#1 } {
  preset  .tl_set:c  = { l__tensorstyles_tensor_#1_local_preset  } ,
  preset* .tl_set:c  = { l__tensorstyles_tensor_#1*_local_preset } ,
  unknown .code:n    = {}
 }
}
```

\c__tensorstyles_tensor_empty_pkg_keys_prop
\c__tensorstyles_tensor_einstein_pkg_keys_prop
\c__tensorstyles_tensor_math_pkg_keys_prop
\c__tensorstyles_tensor_arrow_pkg_keys_prop
\c__tensorstyles_tensor_engineer_pkg_keys_prop
\c__tensorstyles_tensor_bra_pkg_keys_prop
\c__tensorstyles_tensor_ket_pkg_keys_prop
\c__tensorstyles_tensor_full_pkg_keys_prop

Define dictionaries of presets.

The following test files are used for this code: tensorstyles-test-008.

```
\prop_const_from_keyval:Nn \c__tensorstyles_tensor_empty_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = false,
 % Arrows
 arrows = false,
 recursive-arrows = false,
```

```
 % Dirac bra-ket
 bra-ket = false,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_einstein_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = true,
 contra-style = alph,
 covariant-style = alph, %greek, %, alph, arabic
 default-indices = covariant,
 index-marker = \ ,
 collapsed-indices = false,
 specific-indices = {},
 index-offset = 0,
 % covariant-indices = 1,
 % indexOffset = 0,
 % Arrows
 arrows = false,
 recursive-arrows = false,
 % Dirac bra-ket
 % bra-ket = false,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_math_pkg_keys_prop {
 % Font
 font = true,
 font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\Make
 % Einstein index notation
 contra-variant = false,
 % Arrows
 arrows = false,
 recursive-arrows = false,
 % Dirac bra-ket
 bra-ket = false,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_arrow_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = false,
```

```
 % Arrows
 arrows = true,
 arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftrightarrow, 3=\overrightleftrightarrow, 4=
 recursive-arrows = false,
 % Dirac bra-ket
 bra-ket = false,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_engineer_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = false,
 % Arrows
 arrows = false,
 recursive-arrows = true,
 recurs-arrows-cmd =\underline,
 % Dirac bra-ket
 bra-ket = false,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_bra_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = false,
 % Arrows
 arrows = false,
 recursive-arrows = false,
 % Dirac bra-ket
 bra-ket = true,
 scale-var = auto,
 delims-var = \langle\rvert,
 mix-delims = true,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_ket_pkg_keys_prop {
 % Font
 font = false,
 % Einstein index notation
 contra-variant = false,
 % Arrows
 arrows = false,
```

```
 recursive-arrows = false,
 % Dirac bra-ket
 bra-ket = true,
 scale-var = auto,
 delims-var = \lvert\rangle,
 mix-delims = true,
}

\prop_const_from_keyval:Nn \c__tensorstyles_tensor_full_pkg_keys_prop {
 % style-inf=d,
 % style-var = \bold,
 % switch-* = false,
 deduce-var= xX,
 % order = 1,
 % Font
 font = true,
 font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\Make
 % Einstein index notation
 contra-variant = true,
 contra-style = alph,
 covariant-style = greek, %greek, %, alph, arabic
 default-indices = covariant,
 index-marker = \ ,
 collapsed-indices = false,
 specific-indices = {},
 index-offset = 0,
 % covariant-indices = 1,
 % indexOffset = 0,
 % Arrows
 arrows = true,
 arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftrightarrow, 3=\widetilde, 4=\overdoublelef
 recursive-arrows = false,
 recurs-arrows-cmd = \underline,
 % Dirac bra-ket
 bra-ket = true,
 scale-var = auto,
 delims-var = \lbrace\rvert,
 mix-delims = true,
 % style= Dirac(bra-ket), Einstein-contravariant, engineer (underline), physics (double-curv
}
```

(*End of definition for* `\c__tensorstyles_tensor_empty_pkg_keys_prop` *and others.*)

\tensorstyles_tensor_define:Nnn

New: 2025-02-17

#1 : `new/declare/renew/provide` corresponding to the type of command declaration desired.

#2 : `variant` of the command group.

#3 : `macro` for name (given by the user) the command that will be created.

Generic declaration of new function from tensor-type.

The following test files are used for this code: tensorstyles-test-005.

```
% \tensorstyles_tensor_define:Nnn #1=new/declare/renew/provide,  #2=variant, #3=macro
\cs_new_protected:Npn \tensorstyles_tensor_define:Nnn #1 #2 #3 {
 % define the tensor-type of command: \declare {\toto}{ o m }{ code }
 \tl_new:c  { l__tensorstyles_tensor_#2_preset_default_tl }
 \exp_args:Nne #1 { #3 }{ s o m m o !e{\char_generate:nn {`_}{8}^} } {
 %% ##1: star, ##2: options, ##3: tensor order, ##4 variable, ##5: einstein indices, ##6: su
 \group_begin:
  % capture and treat eventual indices given
  \tl_if_in:cnTF { l__tensorstyles_tensor_#2_deduce_var_tl } {##3} {
   % \tl_show:c l__tensorstyles_tensor_#1_deduce_var_tl
   \tl_if_novalue:nTF {##5} { \int_set:Nn \l__tensorstyles_tensor_rank_counter_int 0} {
    \tl_set:Nn \l__tensorstyles_indices_tl {##5}
    \tl_map_inline:Nn \l__tensorstyles_indices_tl {
     \tl_if_single:nTF { ####1 } {
      \token_if_eq_meaning:NNTF ####1 ^{ }{
       \token_if_eq_charcode:NNTF ####1 _{ }{
        \int_incr:N \l__tensorstyles_tensor_rank_counter_int
       }
      }
     }
     {
      \int_incr:N \l__tensorstyles_tensor_rank_counter_int
     }
    }
   }
   \tl_set:Nn \l__tensorstyles_tensor_rank_holder_tl { \int_use:N \l__tensorstyles_tensor_ra
  }{
   \tl_set:Nn \l__tensorstyles_tensor_rank_holder_tl { ##3 }
  }

  % Capture and store presets.
  \tl_if_novalue:nTF { ##2 } {
   \tl_set:cn { l__tensorstyles_tensor_#2_preset_default_tl } { default }
  }{
   \use:c { __tensorstyles_tensor_define_local_preset_keys:n } { #2 }
   \keys_set:nn {tensorstyles/preset/#2} { ##2 }
   \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#2_local_preset  } }

   \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#2*_local_preset } }
   \tensorstyles_local_preset:nVVnn {#2} \l_tmpa_tl \l_tmpb_tl { tensor }
   \tl_set:cn { l__tensorstyles_tensor_#2_preset_default_tl } { pres }
```

__tensorstyles_tensor_no:nnnn

New: 2025-02-17

#1 : `grp-cmd(tensor)_variant` to which the setting is related to.
#2 : `variable` to be displayed.
#3 : `tensor order` (integer) of the variable.
#4 : `indices` only if it has to be specified.

Command to display tensor command, ie apply font, then, arrows, index and delimiters.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_tensor_no:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order, #
\cs_new_protected:Npn \__tensorstyles_tensor_no:nnnn #1 #2 #3 #4 {
 \__tensorstyles_add_delims:nnn {#1} { var } {
  \__tensorstyles_add_einsteinIndex:nnnn {#1} {
   % \__tensorstyles_add_fontCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor ord
   \__tensorstyles_add_recursiveCmd:nnn {#1} {
    \__tensorstyles_add_arrowCmd:nnn {#1} {
     \__tensorstyles_add_fontCmd:nnn {#1} {#2} {#3}
    } {#3}
   } {#3}
  } {#3} {#4}
 }
}
\cs_generate_variant:Nn \__tensorstyles_tensor_no:nnnn { nnen }
```

Command Declaration

\l__tensorstyles_cs_name_tl Local and temporary variable to concatenate the name of the command as a token list.

The following test files are used for this code: tensorstyles-test-005.

```
\tl_new:N \l__tensorstyles_cs_name_tl
\seq_new:N \g__tensorstyles_existing_command_seq
```

(*End of definition for* `\l__tensorstyles_cs_name_tl`.)

\DeclareTensorsCmd

New: 2025-02-17

#1 : `command name` to be defined/declared.

#2 : `keyval` (option) to be given to the new command as a first (non-default) setup.

Command accessible by the users to declare a new command from a command group, with eventually options to be given to this new option.

The following test files are used for this code: tensorstyles-test-005.

```
\DeclareDocumentCommand{\DeclareTensorsCmd}{ m o } {
 % \tensorstyles_addto_globalvar_and_setkeys:NNnnnnnn tl-tensor, macro, inf, keyval, preset,
 \tensorstyles_addto_globalvar_and_setkeys:NNnnn \l__tensorstyles_cs_name_tl #1 {d} { #2 } {

 % \tensorstyles_tensor_define:NVn  declare/renew..., variant, macro
 % DeclareDocumentCommand #1 { #2 } { #3 }
 \tensorstyles_tensor_define:NVn \DeclareDocumentCommand \l__tensorstyles_cs_name_tl { #1 }

 \seq_put_right:Nn \g__tensorstyles_existing_command_seq { #1 }
}
```

Rest of the Code (common)

__tensorstyles_set_scale:nnn

New: 2025-02-17

#1 : `grp-cmd_variant` for which the scaling will be set.

#2 : `scale` the desired scale, eg, auto or none or big or Big...

#3 : `delimiters-name` the name of the category of delimiters to scale.

Set the scale of a pair of delimiters.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_set_scale:nnn #1=grp-cmd_variant, #2=scale, #3=delimiter-name
\cs_new:Npn \__tensorstyles_set_scale:nnn #1 #2 #3 {
 \str_case:nnF { #2 }{
  % if the scale desired is 'auto' or 'none'
  { auto } { \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_auto:nnn
  { none } { \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_none:nnn
 }{
  % otherwise, applied the given scale
  \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_big:nnnn { #2 } }
 }
}
```

Scale, delimiters, evaluation

\g__tensorstyles_last_delim
\g__tensorstyles_follow_up

\g__tensorstyles_follow_up is capturing if the delimiters are followed by another, in order to maybe compress them. \g__tensorstyles_last_delim is capturing the last delimiter seen.

```
% counters and local variables
\bool_new:N \g__tensorstyles_follow_up
\tl_new:N \g__tensorstyles_last_delim
```

(*End of definition for* \g__tensorstyles_last_delim *and* \g__tensorstyles_follow_up.)

__tensorstyles_add_delims:nnn

New: 2025-02-17

#1 : grp-cmd_variant for which the scaling will be set.
#2 : delimiters-name name of the delimiters to add.
#3 : inner-arg inside value between the delimiters.

Apply a pair of delimiters (scaled) around an inner argument.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_add_delims:nnn #1=grp-cmd_variant, #2=delimiter-name, #3=value
\cs_new:Npn \__tensorstyles_add_delims:nnn #1 #2 #3 {
 \bool_if:cTF {l__tensorstyles_tensor_#1_switch_dirac_bool} {
  \tl_use:c { l__tensorstyles_tensor_#1_scale_#2_tl } { tensor_#1 } { #2 } { #3 }
  \tl_set:Nx \l_tmpb_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
  \bool_if:cTF {l__tensorstyles_tensor_#1_mix_delimiter_bool} {
   \seq_map_inline:Nn \g__tensorstyles_existing_command_seq {
    \bool_gset_false:N \g__tensorstyles_follow_up
    \token_if_eq_meaning:NNT \l_peek_token ##1 {
     \bool_gset_true:N \g__tensorstyles_follow_up
     \tl_gset:Nx \g__tensorstyles_last_delim { \tl_item:cn { l__tensorstyles_tensor_#1_delim
     \seq_map_break:
    }
   }
  } {
   \bool_gset_false:N \g__tensorstyles_follow_up
  }
 }{
  #3
 }
}
```

__tensorstyles_scale_auto:nnn

New: 2025-02-17

#1 : `grp-cmd_variant` for which the scaling will be set.

#2 : `delimiters-name` name of the delimiters to add.

#3 : `inner-code` inside value to be displayed between the delimiters.

Command for auto scaling delimiters (mleft and mright).

The following test files are used for this code: tensorstyles-test-005.

```
%\__tensorstyles_scale_auto:nnn #1=grp-cmd_variant, #2=delimiters-name, #3=inner-arg
\cs_new:Npn \__tensorstyles_scale_auto:nnn #1 #2 #3 {
 \tl_set:Nx \l_tmpa_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
 \tl_set:Nx \l_tmpb_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
 \tl_if_empty:NTF \l_tmpa_tl { } {
  \bool_if:NTF \g__tensorstyles_follow_up {
   \bool_if:cTF {l__tensorstyles_#1_mix_delimiter_bool} {
    \tl_if_eq:NNTF \g__tensorstyles_last_delim \l_tmpa_tl {
     \! \__tensorstyles_auto_left:n .
    } {
     \tl_if_eq:NnTF \g__tensorstyles_last_delim { \rvert } {
      \tl_if_eq:NnTF \l_tmpa_tl { \lvert } {
       \! \__tensorstyles_auto_left:n .
      } {
       \__tensorstyles_auto_left:n \l_tmpa_tl
      }
     } {
      \__tensorstyles_auto_left:n \l_tmpa_tl
     }
    }
   }{
    \__tensorstyles_auto_left:n \l_tmpa_tl
   }
  } {
   \__tensorstyles_auto_left:n \l_tmpa_tl
  }
 }
 #3
 \tl_if_empty:NTF \l_tmpb_tl { } {
  \__tensorstyles_auto_right:n \l_tmpb_tl
 }
}
```

__tensorstyles_scale_none:nnn

New: 2025-02-17

#1 : `grp-cmd_variant` for which the scaling will be set.

#2 : `delimiters-name` name of the delimiters to add.

#3 : `inner-code` inside value to be displayed between the delimiters.

Command for intentionally no-scaling delimiters.

The following test files are used for this code: tensorstyles-test-005.

```
%\__tensorstyles_scale_none:nnn #1=grp-cmd_variant, #2=delimiters-name, #3=inner-arg
\cs_new:Npn \__tensorstyles_scale_none:nnn #1 #2 #3 {
 \tl_set:Nx \l_tmpa_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
 \tl_set:Nx \l_tmpb_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
 \tl_if_empty:NTF \l_tmpa_tl { } {
  \bool_if:NTF \g__tensorstyles_follow_up {
   \bool_if:cTF {l__tensorstyles_#1_mix_delimiter_bool} {
    \tl_if_eq:NNTF \g__tensorstyles_last_delim \l_tmpa_tl { } {
     \tl_if_eq:NnTF \g__tensorstyles_last_delim { \rvert } {
      \tl_if_eq:NnTF \l_tmpa_tl { \lvert } { } {
       \__tensorstyles_dont_use_dot:x { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1
      }
     } {
      \__tensorstyles_dont_use_dot:x { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 }
     }
    }
   } {
    \__tensorstyles_dont_use_dot:x { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
   }
  } {
   \__tensorstyles_dont_use_dot:x { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
  }
 }
 #3
 \__tensorstyles_dont_use_dot:x { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
}
```

`\__tensorstyles_dont_use_dot:n`

New: 2025-02-17

`#1` : `delimiter-symbol` to be not scaled.

Command to force no scaling on the given delimiter.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_dont_use_dot:n #1=delimiter-symbol
\cs_new:Npn \__tensorstyles_dont_use_dot:n #1 {
 \str_if_eq:nnF { #1 } { . } { #1 }
}
\cs_generate_variant:Nn \__tensorstyles_dont_use_dot:n { x }
```

__tensorstyles_scale_big:nnnn
__tensorstyles_scale_big_auxi:nNn

New: 2025-02-17

#1 : `scale` the desired scale, eg, big or Big...
#2 : `grp-cmd_variant` for which the scaling will be set.
#3 : `delimiters-name` the name of the category of delimiters to scale.
#4 : `inner-code` inside value to be displayed between the delimiters.

Apply the scale of a pair of delimiters and display what is inside.

The following test files are used for this code: tensorstyles-test-005.

```
%\__tensorstyles_scale_big:nnnn #1 #2=grp-cmd_variant, #3=delimiters-name, #4=inner-code
\cs_new:Npn \__tensorstyles_scale_big:nnnn #1 #2 #3 #4 {
 \__tensorstyles_scale_big_auxi:ncnn { #1 } {l__tensorstyles_#2_delims_#3_tl} { #4 } { #2 }
}
%
% \__tensorstyles_scale_big_auxi:nNn #1=scaling, #2=delimiters, #3=inner-code, #4=grp-cmd
\cs_new:Npn \__tensorstyles_scale_big_auxi:nNnn #1 #2 #3 #4 {
 \tl_set:Nx \l_tmpa_tl { \tl_item:Nn #2 { 1 } }
 \tl_set:Nx \l_tmpb_tl { \tl_item:Nn #2 { 2 } }
 \tl_if_empty:NTF \l_tmpa_tl { } {
  \bool_if:NTF \g__tensorstyles_follow_up {
   \bool_if:cTF {l__tensorstyles_#4_mix_delimiter_bool} {
    \tl_if_eq:NNTF \g__tensorstyles_last_delim \l_tmpa_tl { } {
     \tl_if_eq:NnTF \g__tensorstyles_last_delim { \rvert } {
      \tl_if_eq:NnTF \l_tmpa_tl { \lvert } { } {
       \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
      }
     }{
      \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
     }
    }
   }{
    \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
   }
  }{
   \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
  }
 }
 #3
 \tl_use:c { #1 r } { \tl_item:Nn #2 { 2 } }
}
\cs_generate_variant:Nn \__tensorstyles_scale_big_auxi:nNnn { nc }
```

\nam__tensorstyles_evaluation_sub_sup:nnnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `variant` Name of the variant grp-cmd

#2 : `sub` subscript to add to the inner code

#3 : `sup` superscript to add to the inner code

#4 : `inner code` on which the sub and sup will be added

Function to add sub and superscript to an inner variable.

The following test files are used for this code: tensorstyles-test-005.

```
%
% \__tensorstyles_evaluation_sub_sup:nnnn #1=variant #2=sub #3=sup #4=inner part
\cs_new:Npn \__tensorstyles_evaluation_sub_sup:nnnn #1 #2 #3 #4 {
 \tl_if_novalue:nTF {#2} {
  % if no value, skip sub-point
  \tl_if_novalue:nTF {#3} {
   % if no value skip sup-point, then add only the code-fraction
   #4
  }{
   % if value add only sup-point
   {#4}
   \sp{ #3 }
  }
 }{
  % if value, add sub-point
  {#4}
  \sb{ #2 }
  \tl_if_novalue:nTF {#3} {
   % if no value skip sup-point
  }{
   \sp{ #3 }
  }
 }
}
```

\l__indice_sub_bool `\l__indice_sub_bool` is triggering either indices should be displayed

```
% counters and local variables
\bool_new:N \l__indice_sub_bool
```

(*End of definition for* `\l__indice_sub_bool`.)

\l__tensorstyles_var_int_tmpa
\g__tensorstyles_offset_int_tmpa

\l__tensorstyles_var_int_tmpa is internal index counter \g__tensorstyles_offset_int_tmpa is the offset to apply on the index counter.

```
% counters and local variables
\int_new:N \l__tensorstyles_var_int_tmpa
\int_new:N \g__tensorstyles_offset_int_tmpa
\int_gset:Nn \g__tensorstyles_offset_int_tmpa {0}
```

(*End of definition for* `\l__tensorstyles_var_int_tmpa` *and* `\g__tensorstyles_offset_int_tmpa`.)

\l__tensorstyles_covariant_seq_tmpa
\l__tensorstyles_covariant_manual_seq_tmpa
\l__tensorstyles_contra_seq_tmpa
\l__tensorstyles_contra_manual_seq_tmpa
\l__tensorstyles_indices_tl
\l_tmp_covariant_tl
\l_tmp_contra_tl
\l_tmpc_dim

\l__tensorstyles_covariant_seq_tmpa is local sequence of covariant index \l__tensorstyles_covar
is the local sequence of covariant index given manually \l__tensorstyles_contra_seq_tmpa
is the local sequence of contravariant of index \l__tensorstyles_contra_manual_seq_tmpa
is the local sequence of contravariant of index given manually \l__tensorstyles_indices_tl
is the token list that will be decoded to create the sequence \l_tmp_covariant_tl
\l_tmp_contra_tl \l_tmpc_dim the internal length for collapsed indices

```
% counters and local variables
\seq_new:N \l__tensorstyles_covariant_seq_tmpa
\seq_new:N \l__tensorstyles_covariant_manual_seq_tmpa
\seq_new:N \l__tensorstyles_contra_seq_tmpa
\seq_new:N \l__tensorstyles_contra_manual_seq_tmpa

\tl_new:N \l__tensorstyles_indices_tl
\tl_new:N \l_tmp_covariant_tl
\tl_new:N \l_tmp_contra_tl

\dim_new:N \l_tmpc_dim
```

(*End of definition for* `\l__tensorstyles_covariant_seq_tmpa` *and others. These variables are documented on page* **??**.)

__tensorstyles_add_manual_indices:nn

New: 2025-02-12

Updated: 2025-02-12

#1 : `grp-cmd(tensor) variant` variant of the tensor grp-cmd that is used.

#2 : `index list` to add manually

Adds indices manually (since they have been given by the user) without collapse of indices.

The following test files are used for this code: tensorstyles-test-006.

```
%\__tensorstyles_add_manual_indices:nn #1= grp-cmd(tensor) variant, #2 = index list
\cs_new_protected:Npn \__tensorstyles_add_manual_indices:nn #1 #2 {
 \tl_set:Nn \l__tensorstyles_indices_tl {#2}
 % browse the list
 \tl_map_inline:Nn \l__tensorstyles_indices_tl
 {
  \tl_if_single:nTF { ##1 } {
   \token_if_eq_meaning:NNTF ##1 ^{
    \bool_set_false:N \l__indice_sub_bool
   }{
    \token_if_eq_charcode:NNTF ##1 _{
     \bool_set_true:N \l__indice_sub_bool
    }{
     \settowidth{\l_tmpa_dim}{$##1$}
     \bool_if:NTF \l__indice_sub_bool{
      \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
       \exp_not:n { \mathmakebox[} \dim_use:N \l_tmpa_dim \exp_not:n {][c]}
       { ##1 }
      }
      \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
       \exp_not:n { \mathmakebox[} \dim_use:N \l_tmpa_dim \exp_not:n {][c]}
       { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
      }
     }{
      \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
       \exp_not:n { \mathmakebox[} \dim_use:N \l_tmpa_dim \exp_not:n {][c]}
       { ##1 }
      }
      \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
       \exp_not:n { \mathmakebox[} \dim_use:N \l_tmpa_dim \exp_not:n {][c]}
       { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
      }
     }
     \int_incr:N \l__tensorstyles_var_int_tmpa
    }
   }
  }{
   %Index is a group
   \settowidth{\l_tmpa_dim}{$##1$}
   \bool_if:NTF \l__indice_sub_bool{
```

__tensorstyles_add_manual_collapsed_indices:n

New: 2025-02-12

Updated: 2025-02-12

`#1` : `index list` to add manually

Adds indices manually (since they have been given by the user) with collapse of indices.

The following test files are used for this code: tensorstyles-test-006.

```
%\__tensorstyles_add_manual_collapsed_indices:n #1= index list
\cs_new_protected:Npn \__tensorstyles_add_manual_collapsed_indices:n #1 {
 \tl_set:Nn \l__tensorstyles_indices_tl {#1}
 \tl_map_inline:Nn \l__tensorstyles_indices_tl
 {
  \tl_if_single:nTF { ##1 } {
   \token_if_eq_meaning:NNTF ##1 ^{
    \bool_set_false:N \l__indice_sub_bool
   }{
    \token_if_eq_charcode:NNTF ##1 _{
     \bool_set_true:N \l__indice_sub_bool
    }{
     \bool_if:NTF \l__indice_sub_bool{
      \seq_put_right:Nx \l__tensorstyles_covariant_manual_seq_tmpa { ##1 }
     }{
      \seq_put_right:Nx \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
     }
     \int_incr:N \l__tensorstyles_var_int_tmpa
    }
   }
  }{
   \bool_if:NTF \l__indice_sub_bool{
    \seq_put_right:Nx \l__tensorstyles_covariant_manual_seq_tmpa { ##1 }
   }{
    \seq_put_right:Nx \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
   }
   \int_incr:N \l__tensorstyles_var_int_tmpa
  }
 }
 \int_step_inline:nn {
  \int_max:nn { \seq_count:N \l__tensorstyles_covariant_manual_seq_tmpa } { \seq_count:N \l_
 }{
  \tl_set:Nx \l_tmp_covariant_tl { \seq_item:Nn \l__tensorstyles_covariant_manual_seq_tmpa {
  \tl_set:Nx \l_tmp_contra_tl { \seq_item:Nn \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
  \settowidth{\l_tmpa_dim}{$\l_tmp_covariant_tl$}
  \settowidth{\l_tmpb_dim}{$\l_tmp_contra_tl$}
  \dim_compare:nTF { \l_tmpa_dim > \l_tmpb_dim }{
   \dim_set_eq:NN \l_tmpc_dim \l_tmpa_dim
  }{
   \dim_set_eq:NN \l_tmpc_dim \l_tmpb_dim
  }

```

__tensorstyles_add_one_einsteinIndex:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `grp-cmd(tensor) variant` variant of the tensor grp-cmd that is used.

#2 : `index-style` ie alph, greek or arabic

#3 : `index-type` ie contra or covariant

Adds only one Einstein index.

The following test files are used for this code: tensorstyles-test-006.

```
% \usepackage{mathtools} % for \mathmakebox

% \__tensorstyles_add_one_einsteinIndex:nn #1= grp-cmd(tensor) variant, #2=Indice type #3=Bl
\cs_new_protected:Npn \__tensorstyles_add_one_einsteinIndex:nnn #1 #2 #3 {
 \settowidth{\l_tmpa_dim}{$\use:c { l__tensorstyles_tensor_#1_indexstyle_#2_tl }{\int_eval:n
 \seq_put_right:cx { l__tensorstyles_#2_seq_tmpa } {
  \exp_not:n { \mathmakebox[\l_tmpa_dim][c] }
  { \exp_not:c { l__tensorstyles_tensor_#1_indexstyle_#2_tl }
  { \int_use:N \l__tensorstyles_var_int_tmpa }
  }
 }

 \bool_if:cTF {l__tensorstyles_tensor_#1_collapsed_indices} {
  % We do not add any marker
 }{
  \seq_put_right:cx { l__tensorstyles_#3_seq_tmpa } {
   \exp_not:n { \mathmakebox[\l_tmpa_dim][c] }
   { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
  }
 }
}
```

__tensorstyles_add_einsteinIndex:nnnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `grp-cmd(tensor) variant` variant of the tensor grp-cmd that is used.

#2 : `variable` variable on which the indices will be added

#3 : `tensor order` to determine the number of indices to add.

#4 : `indices` if the list has been explicitly defined by the user.

Adds multiple Einstein indices

The following test files are used for this code: tensorstyles-test-006.

```
% \__tensorstyles_add_einsteinIndex:nnnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor
\cs_new_protected:Npn \__tensorstyles_add_einsteinIndex:nnnn #1 #2 #3 #4 {
 \bool_if:cTF {l__tensorstyles_tensor_#1_switch_einstein_bool} {
  % true switch einstein
  \seq_clear:N \l__tensorstyles_covariant_seq_tmpa
  \seq_clear:N \l__tensorstyles_contra_seq_tmpa
  % Do we raise an error if we do not add #3 indices?
  \tl_if_novalue:nTF {#4} {
   \tl_if_in:NnTF \c__tensorstyles_digits_tl {#3} {
    % The order is a number
    \int_set:Nn \l__tensorstyles_var_int_tmpa { \int_use:c { l__tensorstyles_tensor_#1_index
    \int_step_inline:nn {#3} {
     % increase the counter
     \int_incr:N \l__tensorstyles_var_int_tmpa

     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l__tensorstyles_var_int_tmpa - \int_use:c { l__t
     \clist_if_in:cVTF { l__tensorstyles_tensor_#1_specific_indices_clist } \l_tmpa_tl
     {
      \exp_args:Nne \__tensorstyles_add_one_einsteinIndex:nnn {#1} { \tl_use:c { l__tensorst
     }
     {
      \exp_args:Nne \__tensorstyles_add_one_einsteinIndex:nnn {#1} { \tl_use:c { l__tensorst
     }
     % add this counter (+ the offset) of the sequence of index
    }
   }{
    % if the order is not a number then add it as if it a vector.
    \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
     \__tensorstyles_add_fontCmd:nnn {#1} {#3} {1}
    }
   }
  }{
   %Indices were given manually
   \bool_if:cTF {l__tensorstyles_tensor_#1_collapsed_indices} {
    \__tensorstyles_add_manual_collapsed_indices:n {#4}
   }{
    \__tensorstyles_add_manual_indices:nn {#1} {#4}
   }
   \int_compare:nTF { \l__tensorstyles_var_int_tmpa = #3 } {
```

__tensorstyles_set_index_style:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `grp-cmd(tensor) variant` variant of the tensor grp-cmd that is used.

#2 : `index-type` covariant or contra.

#3 : `index-style` greek, alph or arabic.

Turn the index integer into a string according to the index-style to use.

The following test files are used for this code: tensorstyles-test-006.

```
% \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= index-type, #3= index-style,
\cs_new:Npn \__tensorstyles_set_index_style:nnn #1 #2 #3 {
 \str_case:nnF {#3} {
  { alph  }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_alph:n } }
  { arabic }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_arabic:n }
  { greek  }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_greek:n }
 }{
  \msg_error:nnxx { tensorstyles } { index-style-option-not-defined }
  {#3} {'#3'}
 }
}
```

__tensorstyles_set_index_style:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `grp-cmd(tensor) variant` variant of the tensor grp-cmd that is used.

#2 : `index-type` covariant or contra.

Set the default index type to use.

The following test files are used for this code: tensorstyles-test-006.

```
962
963 % \__tensorstyles_set_index_type:nnn #1= grp-cmd variant, #2= index-type
964 \cs_new:Npn \__tensorstyles_set_index_type:nn #1 #2 {
965  \tl_set:cn { l__tensorstyles_tensor_#1_default_indices_tl } { #2 }
966  \str_case:nnF {#2} {
967   { covariant }{
968    \tl_set:cn { l__tensorstyles_tensor_#1_default_indices_tl } { covariant }
969    \tl_set:cn { l__tensorstyles_tensor_#1_opposed_indices_tl } { contra }
970   }{ contra }{
971    \tl_set:cn { l__tensorstyles_tensor_#1_default_indices_tl } { contra }
972    \tl_set:cn { l__tensorstyles_tensor_#1_opposed_indices_tl } { covariant }
973   }
974  }{
975   \msg_error:nnxx { tensorstyles } { index-type-option-not-defined }
976   {#2} {'#2'}
977  }
978 }
```

__tensorstyles_add_fontCmd:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `variant` grp-cmd(tensor) variant

#2 : `variable` on which the font command will be applied

#3 : `tensor order` to determine which font select.

Apply a font on a variable according to the tensor order.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_add_fontCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order
\cs_new_protected:Npn \__tensorstyles_add_fontCmd:nnn #1 #2 #3 {
 \bool_if:cTF {l__tensorstyles_tensor_#1_switch_font_bool}{
  % true switch font
  \prop_if_in:cnTF {l_tensorstyles_tensor_#1_font_cmd_prop} {#3} {
   % true
   %\exp_args:NNV
   \prop_item:cn {l_tensorstyles_tensor_#1_font_cmd_prop} {#3} {#2}
  }{
   % false
   false\ the\ order\ #3\ is\ not\ present\ in\ font\ option!
  }
 }{
  % false switch font
  #2
 }
}
```

\int_to_greek:n

New: 2025-02-12

Updated: 2025-02-12

#1 : integer to turn into a symbol (string).

Function that returns the greek symbol corresponding to the integer given.

The following test files are used for this code: tensorstyles-test-005.

```
%%
\cs_new:Npn \int_to_greek:n #1 {
 % \int_to_symbols:nnn {⟨int expr ⟩} {⟨total symbols ⟩} {⟨value to symbol mapping ⟩}
 \int_to_symbols:nnn {#1} { 23 } {
  { 1  }{ \alpha }
  { 2  }{ \beta }
  { 3  }{ \gamma }
  { 4  }{ \delta }
  { 5  }{ \varepsilon }
  { 6  }{ \zeta }
  { 7  }{ \eta }
  { 8  }{ \theta }
  { 9  }{ \iota }
  { 10 }{ \kappa }
  { 11 }{ \lambda }
  { 12 }{ \mu }
  { 13 }{ \nu }
  { 14 }{ \xi }
  { 15 }{  o }
  { 16 }{ \pi }
  { 17 }{ \varrho }
  { 18 }{ \sigma }
  { 19 }{ \tau }
  { 20 }{ \upsilon }
  { 21 }{ \phi }
  { 22 }{ \chi }
  { 23 }{ \psi }
  { 23 }{ \omega }
 }
}
```

\recursive_cmd:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `cmd` to apply recursively

#2 : `number` of occurrence to apply

#3 : `variable` on which the recursively command will be applied

Apply a given number of times a command on an input variable. The input command (to apply) can only take one positional argument.

The following test files are used for this code: tensorstyles-test-005.

```
% internal variable to store the counter of occurrence
\int_new:N \g__tensorstyles_int_tmpa
%
% \recursive_cmd:nnn cmd, number, var
\cs_set:Npn \recursive_cmd:nnn #1 #2 #3 {
 \group_begin:
  %
  % all variable assignments will be constrained in this group
  \int_gset:Nn \g__tensorstyles_int_tmpa {#2}
  %
  % while the number of call is higher than 1
  \int_compare:nNnTF {#2} > {1} {
   %true % then apply recursively the command
   \int_gdecr:N \g__tensorstyles_int_tmpa
   \exp_args:Nnx \recursive_cmd:nnV {#1} {\int_use:N \g__tensorstyles_int_tmpa} {#1{#3}}
  }{
   % false %
   \int_compare:nNnTF {#2} = {0} {
    %true
    #3
   }{
    % false % equal 1
    #1{#3}
   }
  }
 \group_end:
}
% variant command for \recursive_cmd:nnn
\cs_generate_variant:Nn \recursive_cmd:nnn { nnV, vnn }
```

__tensorstyles_add_recursiveCmd:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `variant` grp-cmd(tensor) variant

#2 : `variable` on which the recursively command will be applied

#3 : `tensor order` which will correspond to the number of occurrence to apply recursively

Apply (if needed) as many time as the tensor order a command on an input variable.

The following test files are used for this code: tensorstyles-test-005.

```
%
% \__tensorstyles_add_recurseCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor ord
\cs_new_protected:Npn \__tensorstyles_add_recursiveCmd:nnn #1 #2 #3 {
 \bool_if:cTF {l__tensorstyles_tensor_#1_switch_recursiveArrows_bool} {
  % true switch recursive
  \tl_if_in:NnTF \c__tensorstyles_digits_tl {#3} {
   % The order is a number
   \recursive_cmd:vnn {l__tensorstyles_tensor_#1_recursive_arrowCmd_tl} {#3} {#2}
  }{
   % else the order is a letter
   \__tensorstyles_add_arrowCmd:nnn {#1} {#2} {#3}
  }
 }{
  % false switch recursive
  #2
 }
}
```

__tensorstyles_add_arrowCmd:nnn

New: 2025-02-12

Updated: 2025-02-12

#1 : `variant` grp-cmd(tensor) variant

#2 : `variable` on which the arrow command will be applied

#3 : `tensor order` to determine which arrow command apply

Apply an arrow-command according to the tensor-order on an input variable.

The following test files are used for this code: tensorstyles-test-005.

```
% \__tensorstyles_add_arrowCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order
\cs_new_protected:Npn \__tensorstyles_add_arrowCmd:nnn #1 #2 #3 {
 \bool_if:cTF {l__tensorstyles_tensor_#1_switch_arrows_bool} {
  % true switch arrow
  \prop_if_in:cnTF {l_tensorstyles_tensor_#1_arrow_cmd_prop} {#3} {
   % true
   \prop_item:cn {l_tensorstyles_tensor_#1_arrow_cmd_prop} {#3} {#2}
  }{
   % false
   false\ the\ order\ #3\ is\ not\ present\ in\ arrows\ option!
   \prop_show:c {l_tensorstyles_tensor_#1_arrow_cmd_prop}
  }
 }{
  % false switch arrow
  #2
 }
}
```

\MakeBoldUppercase
\MakeUpperBB
\MakeUpperCal
\MakeUpperFrak
\overdoubleleftrightarrow
\overrightleftrightarrow
\overrightdoubleleftrightarrow

New: 2025-02-12

Updated: 2025-02-12

#1 : `cmd` to apply recursively

#2 : `number` of occurrence to apply

#3 : `variable` on which the recursively command will be applied

Apply a given number of times a command on an input variable. The input command (to apply) can only take one positional argument.

The following test files are used for this code: tensorstyles-test-005.

```
%
% \RequirePackage{amsmath}% \overrightarrow
% \RequirePackage{amssymb}% \curvearrowright
% \RequirePackage{bm} % recommended for bold math
%
\DeclareDocumentCommand{\MakeBoldUppercase}{ m }{
 \mathbf{\MakeUppercase{#1}}
}
\DeclareDocumentCommand{\MakeUpperBB}{ m }{
 \mathbb{\MakeUppercase{#1}}
}
\DeclareDocumentCommand{\MakeUpperCal}{ m }{
 \mathcal{\MakeUppercase{#1}}
}
\DeclareDocumentCommand{\MakeUpperFrak}{ m }{
 {\mathfrak{\MakeUppercase{#1}}}
}
\DeclareDocumentCommand{\overdoubleleftrightarrow}{ m }{
 {\overleftrightarrow{\overleftrightarrow{#1}}}
}
\DeclareDocumentCommand{\overrightleftrightarrow}{ m }{
 {\overrightarrow{\overleftrightarrow{#1}}}
}
\DeclareDocumentCommand{\overrightdoubleleftrightarrow}{ m }{
 {\overrightarrow{\overleftrightarrow{\overleftrightarrow{#1}}}}
}
```

Generate variants

__tensorstyles_set_scale:nnn

New: 2025-02-17

#1 : `grp-cmd_variant` for which the scaling will be set.

#2 : `scale` the desired scale, eg, auto or none or big or Big...

#3 : `delimiters-name` the name of the category of delimiters to scale.

Set the scale of a pair of delimiters .

The following test files are used for this code: tensorstyles-test-005.

```
% \tensorstyles_set_keys:nn #1:tensor, #2:keyval
\cs_new_protected:Npn \tensorstyles_set_keys:nn #1 #2 {
 %
 \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_preset  } }
 \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_preset } }
 \use:c { __tensorstyles_tensor_define_local_preset_keys:n } { #1 }
 \keys_set:nn {tensorstyles/preset/#1} { #2 }

 \tl_if_eq:cnF { l__tensorstyles_tensor_#1_local_preset } { None } {
  \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_local_preset } }
  \tl_set:cx { l__tensorstyles_tensor_#1_preset } { \tl_use:c { l__tensorstyles_tensor_#1_lo
 }

 \tl_if_eq:cnF { l__tensorstyles_tensor_#1*_local_preset } { None } {
  \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_local_preset } }
  \tl_set:cx { l__tensorstyles_tensor_#1*_preset } { \tl_use:c { l__tensorstyles_tensor_#1*_
 }

 \seq_if_in:NnTF \l__tensorstyles_tensor_variant_seq { #1 } {
  % True tensor
  \prop_set_eq:cc { l__tensorstyles_tensor_#1_user_keys_prop  } { c__tensorstyles_tensor_ \t
  \prop_set_eq:cc { l__tensorstyles_tensor_#1*_user_keys_prop } { c__tensorstyles_tensor_ \t

  \__tensorstyles_set_default_from_keyval:nVnn { #1  } \l_tmpa_tl { #2 } { tensor }
  \__tensorstyles_set_default_from_keyval:nVnn { #1* } \l_tmpb_tl { #2 } { tensor }
 }{
  % False tensor
  \seq_if_in:NnTF \l__tensorstyles_i_variant_seq { #1 }{
   % True I
   \prop_set_eq:cc { l__tensorstyles_tensor_#1_user_keys_prop  } { c__tensorstyles_i_ \tl_us
   \prop_set_eq:cc { l__tensorstyles_tensor_#1*_user_keys_prop } { c__tensorstyles_i_ \tl_us

   \__tensorstyles_set_default_from_keyval:nVnn { #1 } \l_tmpa_tl { #2 } { i }
   \__tensorstyles_set_default_from_keyval:nVnn { #1* } \l_tmpb_tl { #2 } { i }
  } {
   % False I
   \msg_error:nnxx { tensorstyles } { tensor-not-defined }
   { \token_to_str:N #1  }
   { \token_to_str:N \tensorsset }
  }
 }
}
```

Messages

messages — Set the messages that can be emitted.

New: 2025-02-17

```
% \msg_new:nnnn { tensorstyles } { command-already-defined }
% { Command~'#1'~already~defined! }
% {
%  You~have~used~#2~with~a~command~that~already~has~a~definition. \\
%  The~existing~definition~of~'#1'~will~not~be~altered.
% }
\msg_new:nnn { tensorstyles } { rank_indices_missmatch }
 { The~number~of~given~indices~and~rank~do~not~match~(#1) }
\msg_new:nnn { tensorstyles } { index-type-option-not-define }
 { The~default-indices~is~only~accepting~contra~or~covariant~choice }
\msg_new:nnn { tensorstyles } { index-style-option-not-defined }
 { Index~styles~(ie~contra-style~or~covariant-style)~on~accept~greek,~alph~or~arabic~choice
```

Declaring & Generate new group of commands

\tensorsset{all} — First setup of the options for 'all' commands.

The following test files are used for this code: tensorstyles-test-005.

```
\@ifpackageloaded{mleftright}
{ \tensorsset{all}[scale-auto = mleftmright] }
{ \tensorsset{all}[scale-auto = leftright]   }
```

(*End of definition for* `\tensorsset{all}`. *This function is documented on page* **??**.)

\DeclareTensorsCmd{\testCmd} — Declare the official commands of the package: testCmd

The following test files are used for this code: tensorstyles-test-005.

```
\DeclareTensorsCmd{\tensor}[preset=full]
```

(*End of definition for* `\DeclareTensorsCmd{\testCmd}`. *This function is documented on page* **??**.)

```
\endinput
```

⟨/package⟩

Change History

v0.1.1.2

General: original