

# 1 Commands description for users

```
1 <*package>
2 %\ProvidesExplPackage{tensorstyles}{2026-04-28}{1.0.1.4}{This package provides a generic cus
3 \RequirePackage{expl3}
4 % \RequirePackage{xparse,l3keys2e}% deprecated now.
5 \RequirePackage{amsmath}% \overrightarrow
6 \RequirePackage{amssymb}% \curvearrowright
7 % \RequirePackage{bm}% recommended for bold math
8 \RequirePackage{mathtools}% for \mathmakebox

PACKAGE

\c__tensorstyles_digits_tl Constant variables containing the digits.
9 \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.

10 % Define keys/options for the package.
11 \keys_define:nn { tensorstyles / pkg }
12 {
13 % italic .choice:,
14 % italic .usage:n = { load },
15 % italic .default:n = { true },
16 % italic / true .code:n =
17 % {
18 % \bool_set_true:N \l__tensorstyles_pkg_italic_bool
19 % \bool_set_false:N \l__tensorstyles_pkg_upright_bool
20 % },
21 % italic / false .code:n =
22 % {
23 % \bool_set_false:N \l__tensorstyles_pkg_italic_bool
24 % \bool_set_true:N \l__tensorstyles_pkg_upright_bool
25 % },
26 % italic / unknown .code:n = { \msg_error:nxx { tensorstyles } { boolean-values-only } \l_k
27 % %
28 % upright .choice:,
29 % upright .usage:n = { load },
30 % upright .default:n = { true },
31 % upright / true .code:n =
32 % {
33 % \bool_set_true:N \l__tensorstyles_pkg_upright_bool
34 % \bool_set_false:N \l__tensorstyles_pkg_italic_bool
35 % },
36 % upright / false .code:n =
37 % {
38 % \bool_set_false:N \l__tensorstyles_pkg_upright_bool
39 % \bool_set_true:N \l__tensorstyles_pkg_italic_bool
40 % },
41 % upright / unknown .code:n = { \msg_error:nxx { tensorstyles } { boolean-values-only } \l_
42 % upright .initial:n = true,
43 }
```

```

44 % process the options.
45 \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` 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.  
`\c__tensorstyles_all_pkg_keys_prop` *The following test files are used for this code: tensorstyles-test-003.*

```

46 \prop_const_from_keyval:Nn \c__tensorstyles_all_pkg_keys_prop {scale-auto = leftright}
47 \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.)

```

`\tensorssset{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.*

```

48 % Define the options for all group commands (package included).
49 \keys_define:nn { tensorstyles / all / all } {
50   scale-auto .choice:,
51   scale-auto / leftright .code:n = {
52     \cs_set_eq:NN \__tensorstyles_auto_left:n \left
53     \cs_set_eq:NN \__tensorstyles_auto_right:n \right
54   },
55   scale-auto / mleftmright .code:n = {
56     \cs_set_nopar:Npn \__tensorstyles_auto_left:n { \mleft }
57     \cs_set_nopar:Npn \__tensorstyles_auto_right:n { \mright }
58   }
59 }

```

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

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

```

60 \seq_new:N \l__tensorstyles_new_var_seq

(End of definition for \l__tensorstyles_new_var_seq.)

```

---

`\__tensorstyles_new` #1 : variable-type eg. tl (token list), dim, seq, etc.  
New: 2025-02-17 #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.*

```

61 \cs_new:Npn \__tensorstyles_new:nmmn #1 #2 #3 #4 {
62   \seq_set_from_clist:Nn \l__tensorstyles_new_var_seq { #4 }
63   \seq_map_inline:Nn \l__tensorstyles_new_var_seq
64   { \cs:w #1_new:c \cs_end: { l__tensorstyles_#2_#3_##1_#1 } }
65 }

```

## 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.*

```
66 % \__tensorstyles_set_default_from_keyval: variant, preset, keyval, group command
67 \cs_new_protected:Npn \__tensorstyles_set_default_from_keyval:nnnn #1 #2 #3 #4 {
68 % tensorstyles~set~default~from~keyval
69 % Check if keyvalues have been given.
70 \tl_if_novalue:nTF { #3 }
71 { % no user keys given so _user_keys_prop is taking pkg prop
72 \prop_set_eq:cc { l__tensorstyles_#4_#1_user_keys_prop } { c__tensorstyles_#4_#2_pkg_keys_
73 }{ % user key given and put it in prop
74 \prop_put_from_keyval:cn { l__tensorstyles_#4_#1_user_keys_prop } { #3 }
75 }% end if
76 %
77 % prepare the map of keys
78 \__tensorstyles_set_default_auxi:cnn { l__tensorstyles_#4_#1_user_keys_prop } { #1 } { #4 }
79 %
80 % set this map as default
81 \keys_set:mn { tensorstyles/#4/#1 } { default }
82 }
83 \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.*

```
84 \cs_new_protected:Npn \__tensorstyles_set_default_auxi:Nnn #1 #2 #3 {
85 \__tensorstyles_set_default_auxii:fnn { \prop_to_keyval:N #1 } { #2 } { #3 }
86 }
87 \cs_generate_variant:Nn \__tensorstyles_set_default_auxi:Nnn { c }
88 %
89 % key-val, variant, group command
90 \cs_new_protected:Npn \__tensorstyles_set_default_auxii:nnn #1 #2 #3 {
91 \keys_define:nn { tensorstyles/#3/#2 } { default .meta:n = { #1 } }
92 }
93 \cs_generate_variant:Nn \__tensorstyles_set_default_auxii:nnn { f }
```

---

`\tensorssset` #1 : command for which options will be set or reset.  
#2 : options input given as setting options.

---

New: 2025-02-17

---

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.*

```

94 % \tensorssset m:command:#1 o:options:#2
95 \DeclareDocumentCommand{\tensorssset}{ m o } {
96   \str_if_eq:nnTF { #1 } { all }
97   {
98     % True: if the 'all' commands have to be set.
99     \__tensorstyles_set_default_from_keyval:nnnn { all } {empty} { #2 } { all }
100    \keys_set:nn { tensorstyles/all/all } { default }
101  }{
102    %False: if the command name is given then apply to it:
103    % capture the command name.
104    \tl_set:Nx \l__tensorstyles_tensorssset_tl { \cs_to_str:N #1 }
105    % apply option to the command.
106    \tensorstyles_set_keys:Vn \l__tensorstyles_tensorssset_tl { #2 }
107  }
108 }

```

---

`\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.*

```
109 % tensorstyles_local_keys:nnnn #1=keyval, #2=group-command, #3=command, #4=preset
110 \cs_new_protected:Npn \tensorstyles_local_keys:nnnn #1 #2 #3 #4 {
111 % in tensorstyles-local-keys
112 \tl_set:Nc \l_tmpa_tl { #4 }
113
114 \tl_if_novalue:nTF { #1 }
115 {
116 % keyval NOT given
117 \bool_if:cT { l__tensorstyles_#2_#3_local_keys_bool }
118 {
119 % True
120 \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
121 \bool_set_false:c { l__tensorstyles_#2_#3_local_keys_bool }
122 }
123 }{
124 % keyvalue given
125 \bool_if:cTF { l__tensorstyles_#2_#3_local_keys_bool }
126 {
127 % True
128 \tl_put_right:Nn \l_tmpa_tl { , #1 } % append #1 unexpanded
129 \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
130 }{
131 % False
132 \tl_put_right:Nn \l_tmpa_tl { , #1 } % append #1 unexpanded
133 \keys_set:nV { tensorstyles/#2/#3 } \l_tmpa_tl
134 }
135 \bool_set_true:c { l__tensorstyles_#2_#3_local_keys_bool }
136 }
137 }
138 \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.*

```
139 % tensorstyles_local_preset:nnn #1=group-command, #2=preset, #3=star-preset #4=variant
140 \cs_new_protected:Npn \tensorstyles_local_preset:nVVnn #1 #2 #3 #4 {
141   \tl_if_eq:NnTF { #2 } { None } {
142     \keys_define:nn { tensorstyles/tensor/#1 } { pres .meta:n = { default }}
143   } {
144     \prop_set_eq:cc { l__tensorstyles_#4_#1_user_keys_prop } { c__tensorstyles_#4_#2_pkg_keys_
145     \keys_define:nn { tensorstyles/tensor/#1 } {
146       pres .code:n = {
147         \prop_map_inline:cn { l__tensorstyles_#4_#1_user_keys_prop }
148         { \keys_set:nn { tensorstyles/tensor/#1 } { #####1 = {#####2} } }
149       }
150     }
151   }
152   \tl_if_eq:NnTF { #3 } { None } {
153     \keys_define:nn { tensorstyles/tensor/#1 } { pres* .meta:n = { default }}
154   } {
155     \prop_set_eq:cc { l__tensorstyles_#4_#1*_user_keys_prop } { c__tensorstyles_#4_#3_pkg_keys_
156     \keys_define:nn { tensorstyles/tensor/#1 } {
157       pres* .code:n = {
158         \prop_map_inline:cn { l__tensorstyles_#4_#1*_user_keys_prop }
159         { \keys_set:nn { tensorstyles/tensor/#1 } { #####1* = {#####2} } }
160       }
161     }
162   }
163 }
```

---

`\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`.*

```
164 %
165 % \tensorstyles_addto_globalvar_and_setkeys:NNnnn #1=tl-tensor, #2=macro, #3=inf, #4=keyval,
166 \cs_new_protected:Npn \tensorstyles_addto_globalvar_and_setkeys:NNnnn #1 #2 #3 #4 #5 {
167 % store
168 \tl_set:Nx #1 { \cs_to_str:N #2 }
169
170 % Check if the keyval has values.
171 \tl_if_novalue:nTF { #4 }
172 {
173 % if empty then create the variables by default.
174 \__tensorstyles_preamble_aux:Vnnnn #1 { empty } { empty } { } { #5 }
175 }{
176 % otherwise, create then set the variables.
177 \use:c { __tensorstyles_#5_define_preset_keys:n } { #1 }
178 \keys_set:nn {tensorstyles/preset/#1} { #4 }
179 \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_preset } }
180 \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_preset } }
181
182 \__tensorstyles_preamble_aux:VVVnn #1 \l_tmpa_tl \l_tmpb_tl { #4 } { #5 }
183 % \__tensorstyles_preamble_aux:Veenn #1 { full } { empty } { #4 } { #5 }
184 }
185
186 }
187 %
188 % \__tensorstyles_preamble_aux:nnnn #1=variant, #2=preset, #3=star-preset, #4=key-value, #5=
189 \cs_new_protected:Npn \__tensorstyles_preamble_aux:nnnnn #1 #2 #3 #4 #5 {
190 % Check if the command is in the global sequence
191 \seq_if_in:cnF { l__tensorstyles_#5_variant_seq } { #1 }
192 {
193
194 % if not present
195 % add the command to the sequence
196 \seq_put_left:cn { l__tensorstyles_#5_variant_seq } { #1 }
197 % create new variables (tl) for the new command
198 \use:c { __tensorstyles_#5_variables:n } { #1 }
199 \use:c { __tensorstyles_#5_variables:n } { #1* }
200 % define keys for this command
201 \use:c { __tensorstyles_#5_define_keys:n } { #1 }
202 \use:c { __tensorstyles_#5_define_keys:n } { #1* }
203 }
204
205 % Apply package options
206 \prop_set_eq:cc { l__tensorstyles_#5_#1_user_keys_prop } { c__tensorstyles_#5_#2_pkg_keys_p
207 \prop_set_eq:cc { l__tensorstyles_#5_#1*_user_keys_prop } { c__tensorstyles_#5_#3_pkg_keys_p
208 % Apply the newly given options by default
209 \__tensorstyles_set_default_from_keyval:nnnn { #1* } { #3 } { } { #5 }
210 \__tensorstyles_set_default_from_keyval:nnnn { #1 } { #2 } { #4 } { #5 }
211 }
212 \cs_generate_variant:Nn \__tensorstyles_preamble_aux:nnnnn { VVVnn }
213 \cs_generate_variant:Nn \__tensorstyles_preamble_aux:nnnnn { V }
```

## 'TENSOR' GROUP COMMAND CODE

`\l__tensorstyles_tensor_variant_seq` `\l__tensorstyles_tensor_rank_holder_tl` `\l__tensorstyles_tensor_rank_counter_int` `\l__tensorstyles_tensor_rank_counter_int` 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).

```
214 \seq_new:N \l__tensorstyles_tensor_variant_seq
215 \tl_new:N \l__tensorstyles_tensor_rank_holder_tl
216 \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.*

```
217 % Define variables for the tensor group.
218 % \__tensorstyles_tensor_variables:n variant
219 \cs_new_protected:Npn \__tensorstyles_tensor_variables:n #1 {
220   %%\__tensorstyles_new:nmmm data-type, grp-cmd+variant, category, cs-var-list
221   %
222   % switch
223   \__tensorstyles_new:nmmm { bool } {tensor_#1} { switch } { star }
224   \__tensorstyles_new:nmmm { bool } {tensor_#1} { switch } { font, einstein, dirac, arrows, r
225   % font
226   \__tensorstyles_new:nmmm { prop } {tensor_#1} { font } { cmd }
227   % Einstein
228   \__tensorstyles_new:nmmm { tl } {tensor_#1} { indexstyle } { covariant, contra }
229   % arrows
230   \__tensorstyles_new:nmmm { prop } {tensor_#1} { arrow } { cmd }
231   % recursive arrows
232   \__tensorstyles_new:nmmm { tl } {tensor_#1} { recursive } { arrowCmd }
233   % Dirac bra-ket
234   \__tensorstyles_new:nmmm { tl } {tensor_#1} { scale } { var }
235   \__tensorstyles_new:nmmm { tl } {tensor_#1} { delims } { eval }
236   % keys
237   \__tensorstyles_new:nmmm { prop } {tensor_#1} { user } { keys }
238   \__tensorstyles_new:nmmm { bool } {tensor_#1} { local } { keys }
239 }
```

`\__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.*

```
240 % Define keys/options for tensor group command style.
241 % \__tensorstyles_tensor_define_keys:n variant
242 \cs_new:Npn \__tensorstyles_tensor_define_keys:n #1 {
243   \keys_define:nn { tensorstyles/tensor/#1 } {
```



```

244 switch-* .bool_set:c = { l__tensorstyles_tensor_#1_switch_star_bool },
245 preset .code:n = {},
246 deduce-var .tl_set:c = { l__tensorstyles_tensor_#1_deduce_var_tl },
247
248 % font
249 font .bool_set:c = { l__tensorstyles_tensor_#1_switch_font_bool },
250 font-cmd .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1_font_cmd_prop } },
251
252 % arrow
253 arrows .bool_set:c = { l__tensorstyles_tensor_#1_switch_arrows_bool },
254 arrow-cmd .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1_arrow_cmd_prop } },
255
256 recursive-arrows .bool_set:c = { l__tensorstyles_tensor_#1_switch_recursiveArrows_bool },
257 recurs-arrows-cmd .tl_set:c = { l__tensorstyles_tensor_#1_recursive_arrowCmd_tl },
258
259 % bra-ket
260 bra-ket .bool_set:c = { l__tensorstyles_tensor_#1_switch_dirac_bool },
261 scale-var .choices:nn = { auto, none, big, Big, bigg, Bigg } { \__tensorstyles_set_scale:nn },
262 scale-var .default:n = { auto },
263 scale-var .initial:n = { auto },
264 delims-var .tl_set:c = { l__tensorstyles_tensor_#1_delims_var_tl },
265 mix-delims .bool_set:c = { l__tensorstyles_tensor_#1_mix_delimiter_bool },
266
267 % Einstein
268 contra-variant .bool_set:c = { l__tensorstyles_tensor_#1_switch_einstein_bool },
269 % \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= covariant or contra, #3= op
270 covariant-style .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn },
271 covariant-style .default:n = { greek },
272 covariant-style .initial:n = { greek },
273 contra-style .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn },
274 contra-style .default:n = { alph },
275 contra-style .initial:n = { alph },
276 default-indices .choices:nn = { covariant, contra } { \__tensorstyles_set_index_type:nn },
277 default-indices .default:n = { covariant },
278 default-indices .initial:n = { covariant },
279 index-marker .tl_set:c = { l__tensorstyles_tensor_#1_index_marker_tl },
280 specific-indices .clist_set:c = { l__tensorstyles_tensor_#1_specific_indices_clist },
281 collapsed-indices .bool_set:c = { l__tensorstyles_tensor_#1_collapsed_indices },
282 index-offset .int_set:c = { l__tensorstyles_tensor_#1_index_offset },
283
284 %%% Star Version
285 %%%
286
287 % switch-** .bool_set:c = { l__tensorstyles_tensor_#1*_switch_star_bool },
288 switch-** .code:n = { printSwitchStarStar },
289 preset* .code:n = {},
290 deduce-var* .tl_set:c = { l__tensorstyles_tensor_#1*_deduce_var_tl },
291
292 % font
293 font* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_font_bool },
294 font-cmd* .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1*_font_cmd_prop } },
295
296 % arrow
297 arrows* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_arrows_bool },

```

```

298 arrow-cmd* .code:n = { \prop_set_from_keyval:cn { l_tensorstyles_tensor_#1*_arrow_cmd_prop
299
300 recursive-arrows* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_recursiveArrows_bool },
301 recurs-arrows-cmd* .tl_set:c = { l__tensorstyles_tensor_#1*_recursive_arrowCmd_tl },
302
303 % bra-ket
304 bra-ket* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_dirac_bool },
305 scale-var* .choices:nn = { auto, none, big, Big, bigg, Bigg } { \__tensorstyles_set_scale:
306 delims-var* .tl_set:c = { l__tensorstyles_tensor_#1*_delims_var_tl },
307 mix-delims* .bool_set:c = { l__tensorstyles_tensor_#1*_mix_delimiter_bool },
308
309 % Einstein
310 contra-variant* .bool_set:c = { l__tensorstyles_tensor_#1*_switch_einstein_bool },
311 % \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= covariant or contra, #3= op
312 covariant-style* .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn
313 contra-style* .choices:nn = { alph, arabic, greek } { \__tensorstyles_set_index_style:nnn
314 default-indices* .choices:nn = { covariant, contra } { \__tensorstyles_set_index_type:nn {
315 index-marker* .tl_set:c = { l__tensorstyles_tensor_#1*_index_marker_tl },
316 specific-indices* .clist_set:c = { l__tensorstyles_tensor_#1*_specific_indices_clist },
317 collapsed-indices* .bool_set:c = { l__tensorstyles_tensor_#1*_collapsed_indices },
318 index-offset* .int_set:c = { l__tensorstyles_tensor_#1*_index_offset },
319 }
320 }

```

(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.*

```

321 \cs_new:Npn \__tensorstyles_tensor_define_preset_keys:n #1 {
322 % Initialize variables to "empty" as baseline
323 \tl_set:cn { l__tensorstyles_tensor_#1_preset } { empty }
324 \tl_set:cn { l__tensorstyles_tensor_#1*_preset } { empty }
325 \keys_define:nn { tensorstyles/preset/#1 } {
326   preset .tl_set:c = { l__tensorstyles_tensor_#1_preset } ,
327   preset* .tl_set:c = { l__tensorstyles_tensor_#1*_preset } ,
328   unknown .code:n = {}
329 }
330 }
331
332 \cs_new:Npn \__tensorstyles_tensor_define_local_preset_keys:n #1 {
333 % Initialize variables to "empty" as baseline
334 \tl_set:cn { l__tensorstyles_tensor_#1_local_preset } { None }
335 \tl_set:cn { l__tensorstyles_tensor_#1*_local_preset } { None }
336 \keys_define:nn { tensorstyles/preset/#1 } {
337   preset .tl_set:c = { l__tensorstyles_tensor_#1_local_preset } ,
338   preset* .tl_set:c = { l__tensorstyles_tensor_#1*_local_preset } ,
339   unknown .code:n = {}
340 }
341 }

```

Define dictionaries of presets.

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

```

\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

```

```

342 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_empty_pkg_keys_prop {
343 % Font
344 font = false,
345 % Einstein index notation
346 contra-variant = false,
347 % Arrows
348 arrows = false,
349 recursive-arrows = false,
350 % Dirac bra-ket
351 bra-ket = false,
352 }
353
354 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_einstein_pkg_keys_prop {
355 % Font
356 font = false,
357 % Einstein index notation
358 contra-variant = true,
359 contra-style = alph,
360 covariant-style = alph, %greek, %, alph, arabic
361 default-indices = covariant,

```

```

362 index-marker = \ ,
363 collapsed-indices = false,
364 specific-indices = {},
365 index-offset = 0,
366 % covariant-indices = 1,
367 % indexOffset = 0,
368 % Arrows
369 arrows = false,
370 recursive-arrows = false,
371 % Dirac bra-ket
372 % bra-ket = false,
373 }
374
375 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_math_pkg_keys_prop {
376 % Font
377 font = true,
378 font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\Make
379 % Einstein index notation
380 contra-variant = false,
381 % Arrows
382 arrows = false,
383 recursive-arrows = false,
384 % Dirac bra-ket
385 bra-ket = false,
386 }
387
388 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_arrow_pkg_keys_prop {
389 % Font
390 font = false,
391 % Einstein index notation
392 contra-variant = false,
393 % Arrows
394 arrows = true,
395 arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftarrow, 3=\overrightarrowleftarrow, 4=
396 recursive-arrows = false,
397 % Dirac bra-ket
398 bra-ket = false,
399 }
400
401 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_engineer_pkg_keys_prop {
402 % Font
403 font = false,
404 % Einstein index notation
405 contra-variant = false,
406 % Arrows
407 arrows = false,
408 recursive-arrows = true,
409 recurs-arrows-cmd =\underline,
410 % Dirac bra-ket
411 bra-ket = false,
412 }
413
414 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_bra_pkg_keys_prop {
415 % Font

```

```

416 font = false,
417 % Einstein index notation
418 contra-variant = false,
419 % Arrows
420 arrows = false,
421 recursive-arrows = false,
422 % Dirac bra-ket
423 bra-ket = true,
424 scale-var = auto,
425 delims-var = \langle\rvert,
426 mix-delims = true,
427 }
428
429 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_ket_pkg_keys_prop {
430 % Font
431 font = false,
432 % Einstein index notation
433 contra-variant = false,
434 % Arrows
435 arrows = false,
436 recursive-arrows = false,
437 % Dirac bra-ket
438 bra-ket = true,
439 scale-var = auto,
440 delims-var = \lvert\rangle,
441 mix-delims = true,
442 }
443
444 \prop_const_from_keyval:Nn \c__tensorstyles_tensor_full_pkg_keys_prop {
445 % style-inf=d,
446 % style-var = \bold,
447 % switch-* = false,
448 deduce-var= xX,
449 % order = 1,
450 % Font
451 font = true,
452 font-cmd = {0={}, 1=\boldsymbol, 2=\MakeUppercase, 3=\MakeUpperCal, 4=\MakeUpperBB, 5=\Make
453 % Einstein index notation
454 contra-variant = true,
455 contra-style = alph,
456 covariant-style = greek, %greek, %, alph, arabic
457 default-indices = covariant,
458 index-marker = \ ,
459 collapsed-indices = false,
460 specific-indices = {},
461 index-offset = 0,
462 % covariant-indices = 1,
463 % indexOffset = 0,
464 % Arrows
465 arrows = true,
466 arrow-cmd = {0={}, 1=\overrightarrow, 2=\overleftarrow, 3=\widetilde, 4=\overdoublelef
467 recursive-arrows = false,
468 recurs-arrows-cmd = \underline,
469 % Dirac bra-ket

```

```
470 bra-ket = true,  
471 scale-var = auto,  
472 delims-var = \lbrace\rvert,  
473 mix-delims = true,  
474 % style= Dirac(bra-ket), Einstein-contravariant, engineer (underline), physics (double-curve)  
475 }
```

*(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.*

```
476 % \tensorstyles_tensor_define:Nnn #1=new/declare/renew/provide, #2=variant, #3=macro
477 \cs_new_protected:Npn \tensorstyles_tensor_define:Nnn #1 #2 #3 {
478 % define the tensor-type of command: \declare {\toto}{ o m }{ code }
479 \tl_new:c { l__tensorstyles_tensor_#2_preset_default_tl }
480 \exp_args:Nne #1 { #3 }{ s o m m o !e{\char_generate:nm {'_}{8}^} } {
481 %% ##1: star, ##2: options, ##3: tensor order, ##4 variable, ##5: einstein indices, ##6: su
482 \group_begin:
483 % capture and treat eventual indices given
484 \tl_if_in:cnTF { l__tensorstyles_tensor_#2_deduce_var_tl } {##3} {
485 % \tl_show:c l__tensorstyles_tensor_#1_deduce_var_tl
486 \tl_if_novalue:nTF {##5} { \int_set:Nn \l__tensorstyles_tensor_rank_counter_int 0} {
487 \tl_set:Nn \l__tensorstyles_indices_tl {##5}
488 \tl_map_inline:Nn \l__tensorstyles_indices_tl {
489 \tl_if_single:nTF { ##3#1 } {
490 \token_if_eq_meaning:NNTF #####1 ^{ }{
491 \token_if_eq_charcode:NNTF #####1 _{ }{
492 \int_incr:N \l__tensorstyles_tensor_rank_counter_int
493 }
494 }
495 }
496 {
497 \int_incr:N \l__tensorstyles_tensor_rank_counter_int
498 }
499 }
500 }
501 \tl_set:Nn \l__tensorstyles_tensor_rank_holder_tl { \int_use:N \l__tensorstyles_tensor_ra
502 }{
503 \tl_set:Nn \l__tensorstyles_tensor_rank_holder_tl { ##3 }
504 }
505
506 % Capture and store presets.
507 \tl_if_novalue:nTF { ##2 } {
508 \tl_set:cn { l__tensorstyles_tensor_#2_preset_default_tl } { default }
509 }{
510 \use:c { __tensorstyles_tensor_define_local_preset_keys:n } { #2 }
511 \keys_set:nn {tensorstyles/preset/#2} { ##2 }
512 \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#2_local_preset } }
513
514 \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#2*_local_preset } }
515 \tensorstyles_local_preset:nVnn {#2} \l_tmpa_tl \l_tmpb_tl { tensor }
516 \tl_set:cn { l__tensorstyles_tensor_#2_preset_default_tl } { pres }
517 }
518
519 %% \__tensorstyles_evaluation_sub_sup grp-cmd_variant, sub, sup, code-for-fraction
520 \__tensorstyles_evaluation_sub_sup:nnnn {#2} {##6} {##7} {
521 % if star is present
522 \bool_if:nTF {##1} {
523 % star only
524 % Capture and apply options from keys.
525 \tl_if_novalue:nTF { ##2 } {
526 \tensorstyles_local_keys:nnnn { } { tensor } { #2 } { \tl_use:c {l__tensorstyles_tensor
527 }{
528 \tensorstyles_local_keys:nnnn { ##2 } { tensor } { #2 } { \tl_use:c {l__tensorstyles_te
529 }
530 }
```

---

---

`\_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.*

```
558 % \_tensorstyles_tensor_no:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order, #
559 \cs_new_protected:Npn \_tensorstyles_tensor_no:nnnn #1 #2 #3 #4 {
560 \_tensorstyles_add_delims:nnn {#1} { var } {
561 \_tensorstyles_add_einsteinIndex:nnnn {#1} {
562 % \_tensorstyles_add_fontCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor ord
563 \_tensorstyles_add_recursiveCmd:nnn {#1} {
564 \_tensorstyles_add_arrowCmd:nnn {#1} {
565 \_tensorstyles_add_fontCmd:nnn {#1} {#2} {#3}
566 } {#3}
567 } {#3}
568 } {#3} {#4}
569 }
570 }
571 \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.*

```
572 \tl_new:N \l__tensorstyles_cs_name_tl
573 \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.*

```
574 \DeclareDocumentCommand{\DeclareTensorsCmd}{ m o } {
575 % \tensorstyles_addto_globalvar_and_setkeys:NNnnnnnn tl-tensor, macro, inf, keyval, preset,
576 \tensorstyles_addto_globalvar_and_setkeys:NNnnn \l__tensorstyles_cs_name_tl #1 {d} { #2 } {
577
578 % \tensorstyles_tensor_define:NVn declare/renew..., variant, macro
579 % DeclareDocumentCommand #1 { #2 } { #3 }
580 \tensorstyles_tensor_define:NVn \DeclareDocumentCommand \l__tensorstyles_cs_name_tl { #1 }
581
582 \seq_put_right:Nn \g__tensorstyles_existing_command_seq { #1 }
583 }
```

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.*

```
584 % \__tensorstyles_set_scale:nnn #1=grp-cmd_variant, #2=scale, #3=delimiter-name
585 \cs_new:Npn \__tensorstyles_set_scale:nnn #1 #2 #3 {
586   \str_case:nnF { #2 }{
587     % if the scale desired is 'auto' or 'none'
588     { auto } { \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_auto:nnn
589       { none } { \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_none:nnn
590         }{
591         % otherwise, applied the given scale
592         \tl_set:cn { l__tensorstyles_#1_scale_#3_tl } { \__tensorstyles_scale_big:nnnn { #2 } }
593       }
594     }
```

Scale, delimiters, evaluation

`\g__tensorstyles_last_delim` `\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.

```
595 % counters and local variables
596 \bool_new:N \g__tensorstyles_follow_up
597 \tl_new:N \g__tensorstyles_last_delim
```

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

**#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`.*

```
598 % \__tensorstyles_add_delims:nnn #1=grp-cmd_variant, #2=delimiter-name, #3=value
599 \cs_new:Npn \__tensorstyles_add_delims:nnn #1 #2 #3 {
600   \bool_if:cTF {l__tensorstyles_tensor_#1_switch_dirac_bool} {
601     \tl_use:c { l__tensorstyles_tensor_#1_scale_#2_tl } { tensor_#1 } { #2 } { #3 }
602     \tl_set:Nx \l_tmpb_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
603     \bool_if:cTF {l__tensorstyles_tensor_#1_mix_delimiter_bool} {
604       \seq_map_inline:Nn \g__tensorstyles_existing_command_seq {
605         \bool_gset_false:N \g__tensorstyles_follow_up
606         \token_if_eq_meaning:NNT \l_peek_token ##1 {
607           \bool_gset_true:N \g__tensorstyles_follow_up
608           \tl_gset:Nx \g__tensorstyles_last_delim { \tl_item:cn { l__tensorstyles_tensor_#1_delim
609             \seq_map_break:
610           }
611         }
612       } {
613         \bool_gset_false:N \g__tensorstyles_follow_up
614       }
615     }{
616       #3
617     }
618   }
```

**#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`.*

```
619 %\_tensorstyles_scale_auto:nnn #1=grp-cmd_variant, #2=delimiters-name, #3=inner-arg
620 \cs_new:Npn \_tensorstyles_scale_auto:nnn #1 #2 #3 {
621   \tl_set:Nx \l_tmpa_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
622   \tl_set:Nx \l_tmpb_tl { \tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
623   \tl_if_empty:NTF \l_tmpa_tl { } {
624     \bool_if:NTF \g__tensorstyles_follow_up {
625       \bool_if:cTF {l__tensorstyles_#1_mix_delimiter_bool} {
626         \tl_if_eq:NNTF \g__tensorstyles_last_delim \l_tmpa_tl {
627           \! \_tensorstyles_auto_left:n .
628         } {
629           \tl_if_eq:NnTF \g__tensorstyles_last_delim { \rvert } {
630             \tl_if_eq:NnTF \l_tmpa_tl { \lvert } {
631               \! \_tensorstyles_auto_left:n .
632             } {
633               \_tensorstyles_auto_left:n \l_tmpa_tl
634             }
635           } {
636             \_tensorstyles_auto_left:n \l_tmpa_tl
637           }
638         }
639       }{
640         \_tensorstyles_auto_left:n \l_tmpa_tl
641       }
642     } {
643       \_tensorstyles_auto_left:n \l_tmpa_tl
644     }
645   }
646   #3
647   \tl_if_empty:NTF \l_tmpb_tl { } {
648     \_tensorstyles_auto_right:n \l_tmpb_tl
649   }
650 }
```

---

---

**\\\_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.*

```
651 %\\_tensorstyles_scale_none:nnn #1=grp-cmd_variant, #2=delimiters-name, #3=inner-arg
652 \\cs_new:Npn \\_tensorstyles_scale_none:nnn #1 #2 #3 {
653   \\tl_set:Nx \\l_tmpa_tl { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
654   \\tl_set:Nx \\l_tmpb_tl { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
655   \\tl_if_empty:NTF \\l_tmpa_tl { } {
656     \\bool_if:NTF \\g__tensorstyles_follow_up {
657       \\bool_if:cTF {l__tensorstyles_#1_mix_delimiter_bool} {
658         \\tl_if_eq:NNTF \\g__tensorstyles_last_delim \\l_tmpa_tl { } {
659           \\tl_if_eq:NnTF \\g__tensorstyles_last_delim { \\rvert } {
660             \\tl_if_eq:NnTF \\l_tmpa_tl { \\lvert } { } {
661               \\_tensorstyles_dont_use_dot:x { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 }
662             }
663           } {
664             \\_tensorstyles_dont_use_dot:x { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 }
665           }
666         }
667       } {
668         \\_tensorstyles_dont_use_dot:x { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
669       }
670     } {
671       \\_tensorstyles_dont_use_dot:x { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 1 } }
672     }
673   }
674   #3
675   \\_tensorstyles_dont_use_dot:x { \\tl_item:cn { l__tensorstyles_#1_delims_#2_tl } { 2 } }
676 }
```

---

---

**\\\_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.*

```
677 % \\_tensorstyles_dont_use_dot:n #1=delimiter-symbol
678 \\cs_new:Npn \\_tensorstyles_dont_use_dot:n #1 {
679   \\str_if_eq:nnF { #1 } { . } { #1 }
680 }
681 \\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.*

```

682 %\__tensorstyles_scale_big:nnnn #1 #2=grp-cmd_variant, #3=delimiters-name, #4=inner-code
683 \cs_new:Npn \__tensorstyles_scale_big:nnnn #1 #2 #3 #4 {
684   \__tensorstyles_scale_big_auxi:ncnn { #1 } {l__tensorstyles_#2_delims_#3_tl} { #4 } { #2 }
685 }
686 %
687 % \__tensorstyles_scale_big_auxi:nNn #1=scaling, #2=delimiters, #3=inner-code, #4=grp-cmd
688 \cs_new:Npn \__tensorstyles_scale_big_auxi:nNnn #1 #2 #3 #4 {
689   \tl_set:Nx \l_tmpa_tl { \tl_item:Nn #2 { 1 } }
690   \tl_set:Nx \l_tmpb_tl { \tl_item:Nn #2 { 2 } }
691   \tl_if_empty:NTF \l_tmpa_tl { } {
692     \bool_if:NTF \g__tensorstyles_follow_up {
693       \bool_if:cTF {l__tensorstyles_#4_mix_delimiter_bool} {
694         \tl_if_eq:NNTF \g__tensorstyles_last_delim \l_tmpa_tl { } {
695           \tl_if_eq:NnTF \g__tensorstyles_last_delim { \rvert } {
696             \tl_if_eq:NnTF \l_tmpa_tl { \lvert } { } {
697               \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
698             }
699           }{
700             \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
701           }
702         }
703       }{
704         \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
705       }
706     }{
707       \tl_use:c { #1 l } { \tl_item:Nn #2 { 1 } }
708     }
709   }
710   #3
711   \tl_use:c { #1 r } { \tl_item:Nn #2 { 2 } }
712 }
713 \cs_generate_variant:Nn \__tensorstyles_scale_big_auxi:nNnn { nc }

```

---

`\nam\_tensorstyles_evaluation_sub_sup:nmm`

---

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.*

```
714 %
715 % \_tensorstyles_evaluation_sub_sup:nmm #1=variant #2=sub #3=sup #4=inner part
716 \cs_new:Npn \_tensorstyles_evaluation_sub_sup:nmm #1 #2 #3 #4 {
717   \tl_if_novalue:nTF {#2} {
718     % if no value, skip sub-point
719     \tl_if_novalue:nTF {#3} {
720       % if no value skip sup-point, then add only the code-fraction
721       #4
722     }{
723       % if value add only sup-point
724       {#4}
725       \sp{ #3 }
726     }
727   }{
728     % if value, add sub-point
729     {#4}
730     \sb{ #2 }
731     \tl_if_novalue:nTF {#3} {
732       % if no value skip sup-point
733     }{
734       \sp{ #3 }
735     }
736   }
737 }
```

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

```
738 % counters and local variables
739 \bool_new:N \l__indice_sub_bool
```

*(End of definition for \l\_\_indice\_sub\_bool.)*

`\l_tensorstyles_var_int_tmpa` `\l_tensorstyles_var_int_tmpa` is internal index counter `\g_tensorstyles_offset_int_tmpa`  
`\g_tensorstyles_offset_int_tmpa` is the offset to apply on the index counter.

```
740 % counters and local variables
741 \int_new:N \l_tensorstyles_var_int_tmpa
742 \int_new:N \g_tensorstyles_offset_int_tmpa
743 \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_seq_tmpa` is local sequence of covariant index `\l_tensorstyles_covar`  
`\l_tensorstyles_covariant_manual_seq_tmpa` is the local sequence of covariant index given manually `\l_tensorstyles_contra_seq_tmpa`  
`\l_tensorstyles_contra_seq_tmpa` is the local sequence of contravariant of index `\l_tensorstyles_contra_manual_seq_tmpa`

```
\l_tensorstyles_contra_manual_seq_tmpa
\l_tensorstyles_indices_tl
\l_tmp_covariant_tl
\l_tmp_contra_tl
\l_tmpc_dim
```

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

```
744 % counters and local variables
745 \seq_new:N \l__tensorstyles_covariant_seq_tmpa
746 \seq_new:N \l__tensorstyles_covariant_manual_seq_tmpa
747 \seq_new:N \l__tensorstyles_contra_seq_tmpa
748 \seq_new:N \l__tensorstyles_contra_manual_seq_tmpa
749
750 \tl_new:N \l__tensorstyles_indices_tl
751 \tl_new:N \l_tmp_covariant_tl
752 \tl_new:N \l_tmp_contra_tl
753
754 \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 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`.*

```
755
756 %\__tensorstyles_add_manual_indices:nn #1= grp-cmd(tensor) variant, #2 = index list
757 \cs_new_protected:Npn \__tensorstyles_add_manual_indices:nn #1 #2 {
758   \tl_set:Nn \l__tensorstyles_indices_tl {#2}
759   % browse the list
760   \tl_map_inline:Nn \l__tensorstyles_indices_tl
761   {
762     \tl_if_single:NTF { ##1 } {
763       \token_if_eq_meaning:NNTF ##1 ~{
764         \bool_set_false:N \l__indice_sub_bool
765       }{
766         \token_if_eq_charcode:NNTF ##1 _{
767           \bool_set_true:N \l__indice_sub_bool
768         }{
769           \settowidth{\l_tmpa_dim}{##1$}
770           \bool_if:NTF \l__indice_sub_bool{
771             \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
772               \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
773               { ##1 }
774             }
775             \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
776               \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
777               { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
778             }
779           }{
780             \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
781               \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
782               { ##1 }
783             }
784             \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
785               \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
786               { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
787             }
788           }
789           \int_incr:N \l__tensorstyles_var_int_tmpa
790         }
791       }
792     }{
793       %Index is a group
794       \settowidth{\l_tmpa_dim}{##1$}
795       \bool_if:NTF \l__indice_sub_bool{
796         \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
797           \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
798           { \exp_not:n {##1} }
799         }
800         \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
801           \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
802           \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl }
803         }
804       }{
805         \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
806           \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpa_dim \exp_not:n {} [c]}
807           { \exp_not:n {##1} }
808         }
809         \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
```



---

`\__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.*

```
818
819 %\__tensorstyles_add_manual_collapsed_indices:n #1= index list
820 \cs_new_protected:Npn \__tensorstyles_add_manual_collapsed_indices:n #1 {
821   \tl_set:Nn \l__tensorstyles_indices_tl {#1}
822   \tl_map_inline:Nn \l__tensorstyles_indices_tl
823   {
824     \tl_if_single:nTF { ##1 } {
825       \token_if_eq_meaning:NNTF ##1 ^{
826         \bool_set_false:N \l__indice_sub_bool
827       }{
828         \token_if_eq_charcode:NNTF ##1 _{
829           \bool_set_true:N \l__indice_sub_bool
830         }{
831           \bool_if:NTF \l__indice_sub_bool{
832             \seq_put_right:Nx \l__tensorstyles_covariant_manual_seq_tmpa { ##1 }
833           }{
834             \seq_put_right:Nx \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
835           }
836           \int_incr:N \l__tensorstyles_var_int_tmpa
837         }
838       }
839     }{
840       \bool_if:NTF \l__indice_sub_bool{
841         \seq_put_right:Nx \l__tensorstyles_covariant_manual_seq_tmpa { ##1 }
842       }{
843         \seq_put_right:Nx \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
844       }
845       \int_incr:N \l__tensorstyles_var_int_tmpa
846     }
847   }
848   \int_step_inline:nn {
849     \int_max:nn { \seq_count:N \l__tensorstyles_covariant_manual_seq_tmpa } { \seq_count:N \l__
850   }{
851     \tl_set:Nx \l_tmp_covariant_tl { \seq_item:Nn \l__tensorstyles_covariant_manual_seq_tmpa {
852     \tl_set:Nx \l_tmp_contra_tl { \seq_item:Nn \l__tensorstyles_contra_manual_seq_tmpa { ##1 }
853     \settowidth{\l_tmpa_dim}{\l_tmp_covariant_tl}
854     \settowidth{\l_tmpb_dim}{\l_tmp_contra_tl}
855     \dim_compare:nTF { \l_tmpa_dim > \l_tmpb_dim }{
856       \dim_set_eq:NN \l_tmpc_dim \l_tmpa_dim
857     }{
858       \dim_set_eq:NN \l_tmpc_dim \l_tmpb_dim
859     }
860
861     \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
862     \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpc_dim \exp_not:n {} [c]}
863     { \exp_not:V \l_tmp_covariant_tl }
864   }
865
866     \seq_put_right:Nx \l__tensorstyles_contra_seq_tmpa {
867     \exp_not:n { \mathmakebox[] \dim_use:N \l_tmpc_dim \exp_not:n {} [c]}
868     { \exp_not:V \l_tmp_contra_tl }
869   }
870 }
871 }
```

---

---

**\\\_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.*

```
872 % \usepackage{mathtools} % for \mathmakebox
873
874 % \\_tensorstyles_add_one_einsteinIndex:nn #1= grp-cmd(tensor) variant, #2=Indice type #3=Bl
875 \cs_new_protected:Npn \\_tensorstyles_add_one_einsteinIndex:nnn #1 #2 #3 {
876 \settowidth{\l_tmpa_dim}{\use:c { l__tensorstyles_tensor_#1_indexstyle_#2_tl }}{\int_eval:n
877 \seq_put_right:cx { l__tensorstyles_#2_seq_tmpa } {
878 \exp_not:n { \mathmakebox[\l_tmpa_dim][c] }
879 { \exp_not:c { l__tensorstyles_tensor_#1_indexstyle_#2_tl }
880 { \int_use:N \l__tensorstyles_var_int_tmpa }
881 }
882 }
883
884 \bool_if:cTF {l__tensorstyles_tensor_#1_collapsed_indices} {
885 % We do not add any marker
886 }{
887 \seq_put_right:cx { l__tensorstyles_#3_seq_tmpa } {
888 \exp_not:n { \mathmakebox[\l_tmpa_dim][c] }
889 { \exp_not:v { l__tensorstyles_tensor_#1_index_marker_tl } }
890 }
891 }
892 }
```

---

---

`\__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`.*

```
893
894 % \__tensorstyles_add_einsteinIndex:nnnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor
895 \cs_new_protected:Npn \__tensorstyles_add_einsteinIndex:nnnn #1 #2 #3 #4 {
896   \bool_if:cTF {l__tensorstyles_tensor_#1_switch_einstein_bool} {
897     % true switch einstein
898     \seq_clear:N \l__tensorstyles_covariant_seq_tmpa
899     \seq_clear:N \l__tensorstyles_contra_seq_tmpa
900     % Do we raise an error if we do not add #3 indices?
901     \tl_if_novalue:nTF {#4} {
902       \tl_if_in:NnTF \c__tensorstyles_digits_tl {#3} {
903         % The order is a number
904         \int_set:Nn \l__tensorstyles_var_int_tmpa { \int_use:c { l__tensorstyles_tensor_#1_index
905           \int_step_inline:nn {#3} {
906             % increase the counter
907             \int_incr:N \l__tensorstyles_var_int_tmpa
908
909             \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l__tensorstyles_var_int_tmpa - \int_use:c { l__t
910             \clist_if_in:cVTF { l__tensorstyles_tensor_#1_specific_indices_clist } \l_tmpa_tl
911             {
912               \exp_args:Nne \__tensorstyles_add_one_einsteinIndex:nnn {#1} { \tl_use:c { l__tensorst
913             }
914             {
915               \exp_args:Nne \__tensorstyles_add_one_einsteinIndex:nnn {#1} { \tl_use:c { l__tensorst
916             }
917             % add this counter (+ the offset) of the sequence of index
918             }
919           }{
920             % if the order is not a number then add it as if it a vector.
921             \seq_put_right:Nx \l__tensorstyles_covariant_seq_tmpa {
922               \__tensorstyles_add_fontCmd:nnn {#1} {#3} {1}
923             }
924           }
925         }{
926           %Indices were given manually
927           \bool_if:cTF {l__tensorstyles_tensor_#1_collapsed_indices} {
928             \__tensorstyles_add_manual_collapsed_indices:n {#4}
929           }{
930             \__tensorstyles_add_manual_indices:nn {#1} {#4}
931           }
932           \int_compare:nTF { \l__tensorstyles_var_int_tmpa = #3 } {
933             %Rank and number of indices match
934           }{
935             %User has given too many or not enough indices
936             \msg_error:nnx { tensorstyles } { rank_indices_mismatch } { \msg_line_context: }
937           }
938         }
939         % print
940         #2\sb{
941           \seq_use:Nn \l__tensorstyles_covariant_seq_tmpa {}
942         }\sp{
943           \seq_use:Nn \l__tensorstyles_contra_seq_tmpa {}
944         }
945       }{
946         % false switch font
```

---

---

**`\__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`.*

```
950
951 % \__tensorstyles_set_index_style:nnn #1= grp-cmd variant, #2= index-type, #3= index-style,
952 \cs_new:Npn \__tensorstyles_set_index_style:nnn #1 #2 #3 {
953   \str_case:nnF {#3} {
954     { alph }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_alph:n } }
955     { arabic }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_arabic:n } }
956     { greek }{ \tl_set:cn { l__tensorstyles_tensor_#1_indexstyle_#2_tl } { \int_to_greek:n } }
957   }{
958     \msg_error:nnxx { tensorstyles } { index-style-option-not-defined }
959     {#3} {'#3'}
960   }
961 }
```

---

---

**`\__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.*

```
979
980 % \_tensorstyles_add_fontCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order
981 \cs_new_protected:Npn \_tensorstyles_add_fontCmd:nnn #1 #2 #3 {
982   \bool_if:cTF {l\_tensorstyles_tensor\_#1\_switch\_font\_bool}{
983     % true switch font
984     \prop_if_in:cnTF {l\_tensorstyles_tensor\_#1\_font\_cmd\_prop} {#3} {
985       % true
986       %\exp_args:NNV
987       \prop_item:cn {l\_tensorstyles_tensor\_#1\_font\_cmd\_prop} {#3} {#2}
988     }{
989       % false
990       false\ the\ order\ #3\ is\ not\ present\ in\ font\ option!
991     }
992   }{
993     % false switch font
994     #2
995   }
996 }
```

---

`\int_to_greek:n`

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

New: 2025-02-12

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

Updated: 2025-02-12

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

---

```
997 %%
998 \cs_new:Npn \int_to_greek:n #1 {
999 % \int_to_symbols:nnn {<int expr >} {<total symbols >} {<value to symbol mapping >}
1000 \int_to_symbols:nnn {#1} { 23 } {
1001 { 1 }{ \alpha }
1002 { 2 }{ \beta }
1003 { 3 }{ \gamma }
1004 { 4 }{ \delta }
1005 { 5 }{ \varepsilon }
1006 { 6 }{ \zeta }
1007 { 7 }{ \eta }
1008 { 8 }{ \theta }
1009 { 9 }{ \iota }
1010 { 10 }{ \kappa }
1011 { 11 }{ \lambda }
1012 { 12 }{ \mu }
1013 { 13 }{ \nu }
1014 { 14 }{ \xi }
1015 { 15 }{ o }
1016 { 16 }{ \pi }
1017 { 17 }{ \varrho }
1018 { 18 }{ \sigma }
1019 { 19 }{ \tau }
1020 { 20 }{ \upsilon }
1021 { 21 }{ \phi }
1022 { 22 }{ \chi }
1023 { 23 }{ \psi }
1024 { 23 }{ \omega }
1025 }
1026 }
```

---

`\recursive_cmd:nnn` #1 : cmd to apply recursively  
 #2 : number of occurrence to apply  
 #3 : variable on which the recursively command will be applied

---

New: 2025-02-12  
 Updated: 2025-02-12

---

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.*

```

1027
1028 % internal variable to store the counter of occurrence
1029 \int_new:N \g__tensorstyles_int_tmpa
1030 %
1031 % \recursive_cmd:nnn cmd, number, var
1032 \cs_set:Npn \recursive_cmd:nnn #1 #2 #3 {
1033   \group_begin:
1034   %
1035   % all variable assignments will be constrained in this group
1036   \int_gset:Nn \g__tensorstyles_int_tmpa {#2}
1037   %
1038   % while the number of call is higher than 1
1039   \int_compare:nNnTF {#2} > {1} {
1040     %true % then apply recursively the command
1041     \int_gdecr:N \g__tensorstyles_int_tmpa
1042     \exp_args:Nnx \recursive_cmd:nnV {#1} {\int_use:N \g__tensorstyles_int_tmpa} {#1{#3}}
1043   }{
1044     % false %
1045     \int_compare:nNnTF {#2} = {0} {
1046       %true
1047       #3
1048     }{
1049       % false % equal 1
1050       #1{#3}
1051     }
1052   }
1053 \group_end:
1054 }
1055 % variant command for \recursive_cmd:nnn
1056 \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.*

```
1057 %
1058 % \_tensorstyles_add_recurseCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor ord
1059 \cs_new_protected:Npn \_tensorstyles_add_recursiveCmd:nnn #1 #2 #3 {
1060 \bool_if:cTF {l\_tensorstyles_tensor\_#1\_switch\_recursiveArrows\_bool} {
1061   % true switch recursive
1062   \tl_if_in:NnTF \c\_tensorstyles_digits\_tl {#3} {
1063     % The order is a number
1064     \recursive\_cmd:vnn {l\_tensorstyles\_tensor\_#1\_recursive\_arrowCmd\_tl} {#3} {#2}
1065   }{
1066     % else the order is a letter
1067     \_tensorstyles_add\_arrowCmd:nnn {#1} {#2} {#3}
1068   }
1069 }{
1070 % false switch recursive
1071 #2
1072 }
1073 }
```



---

**\\_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.*

```
1074
1075 % \_tensorstyles_add_arrowCmd:nnn #1= grp-cmd(tensor) variant, #2=variable, #3=tensor order
1076 \cs_new_protected:Npn \_tensorstyles_add_arrowCmd:nnn #1 #2 #3 {
1077   \bool_if:cTF {l\_tensorstyles_tensor\_#1\_switch\_arrows\_bool} {
1078     % true switch arrow
1079     \prop_if_in:cnTF {l\_tensorstyles_tensor\_#1\_arrow\_cmd\_prop} {#3} {
1080       % true
1081       \prop_item:cn {l\_tensorstyles_tensor\_#1\_arrow\_cmd\_prop} {#3} {#2}
1082     }{
1083       % false
1084       false\ the\ order\ #3\ is\ not\ present\ in\ arrows\ option!
1085       \prop_show:c {l\_tensorstyles_tensor\_#1\_arrow\_cmd\_prop}
1086     }
1087   }{
1088     % false switch arrow
1089     #2
1090   }
1091 }
1092
```

---

```

\MakeBoldUppercase
\MakeUpperBB
\MakeUpperCal
\MakeUpperFrak
\overdoubleleftrightharpoon
\overrightleftrightharpoon
\overrightdoubleleftrightharpoon

```

---

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.*

```

1093 %
1094 % \RequirePackage{amsmath}% \overrightarrow
1095 % \RequirePackage{amssymb}% \curvearrowright
1096 % \RequirePackage{bm} % recommended for bold math
1097 %
1098 \DeclareDocumentCommand{\MakeBoldUppercase}{ m }{
1099   \mathbf{\MakeUppercase{#1}}
1100 }
1101 \DeclareDocumentCommand{\MakeUpperBB}{ m }{
1102   \mathbb{\MakeUppercase{#1}}
1103 }
1104 \DeclareDocumentCommand{\MakeUpperCal}{ m }{
1105   \mathcal{\MakeUppercase{#1}}
1106 }
1107 \DeclareDocumentCommand{\MakeUpperFrak}{ m }{
1108   {\mathfrak{\MakeUppercase{#1}}}
1109 }
1110 \DeclareDocumentCommand{\overdoubleleftrightharpoon}{ m }{
1111   {\overleftrightharpoon{\overleftrightharpoon{#1}}}
1112 }
1113 \DeclareDocumentCommand{\overrightleftrightharpoon}{ m }{
1114   {\overrightarrow{\overleftrightharpoon{#1}}}
1115 }
1116 \DeclareDocumentCommand{\overrightdoubleleftrightharpoon}{ m }{
1117   {\overrightarrow{\overleftrightharpoon{\overleftrightharpoon{#1}}}}
1118 }

```

Generate variants

**#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.*

```
1119 % \tensorstyles_set_keys:nn #1:tensor, #2:keyval
1120 \cs_new_protected:Npn \tensorstyles_set_keys:nn #1 #2 {
1121 %
1122 \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_preset } }
1123 \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_preset } }
1124 \use:c { __tensorstyles_tensor_define_local_preset_keys:n } { #1 }
1125 \keys_set:nn {tensorstyles/preset/#1} { #2 }
1126
1127 \tl_if_eq:cnF { l__tensorstyles_tensor_#1_local_preset } { None } {
1128   \tl_set:Nx \l_tmpa_tl { \tl_use:c { l__tensorstyles_tensor_#1_local_preset } }
1129   \tl_set:cx { l__tensorstyles_tensor_#1_preset } { \tl_use:c { l__tensorstyles_tensor_#1_lo
1130 }
1131
1132 \tl_if_eq:cnF { l__tensorstyles_tensor_#1*_local_preset } { None } {
1133   \tl_set:Nx \l_tmpb_tl { \tl_use:c { l__tensorstyles_tensor_#1*_local_preset } }
1134   \tl_set:cx { l__tensorstyles_tensor_#1*_preset } { \tl_use:c { l__tensorstyles_tensor_#1*_
1135 }
1136
1137 \seq_if_in:NnTF \l__tensorstyles_tensor_variant_seq { #1 } {
1138   % True tensor
1139   \prop_set_eq:cc { l__tensorstyles_tensor_#1_user_keys_prop } { c__tensorstyles_tensor_ \t
1140   \prop_set_eq:cc { l__tensorstyles_tensor_#1*_user_keys_prop } { c__tensorstyles_tensor_ \t
1141
1142   \__tensorstyles_set_default_from_keyval:nVnn { #1 } \l_tmpa_tl { #2 } { tensor }
1143   \__tensorstyles_set_default_from_keyval:nVnn { #1* } \l_tmpb_tl { #2 } { tensor }
1144 }{
1145   % False tensor
1146   \seq_if_in:NnTF \l__tensorstyles_i_variant_seq { #1 }{
1147     % True I
1148     \prop_set_eq:cc { l__tensorstyles_tensor_#1_user_keys_prop } { c__tensorstyles_i_ \tl_us
1149     \prop_set_eq:cc { l__tensorstyles_tensor_#1*_user_keys_prop } { c__tensorstyles_i_ \tl_us
1150
1151     \__tensorstyles_set_default_from_keyval:nVnn { #1 } \l_tmpa_tl { #2 } { i }
1152     \__tensorstyles_set_default_from_keyval:nVnn { #1* } \l_tmpb_tl { #2 } { i }
1153 } {
1154   % False I
1155   \msg_error:nxxx { tensorstyles } { tensor-not-defined }
1156   { \token_to_str:N #1 }
1157   { \token_to_str:N \tensorssset }
1158 }
1159 }
1160 }
1161 \cs_generate_variant:Nn \tensorstyles_set_keys:nn { V }
1162 % \cs_generate_variant:Nn \__tensorstyles_set_rubber_length:Nn { c }
1163 % \cs_generate_variant:Nn \__tensorstyles_var_bool_seq:NNN { c }
```

## Messages

---

**messages** Set the messages that can be emitted.

---

New: 2025-02-17

```
1164 % \msg_new:nnnn { tensorstyles } { command-already-defined }
1165 % { Command~'#1'~already~defined! }
1166 % {
1167 % You~have~used~#2~with~a~command~that~already~has~a~definition. \\
1168 % The~existing~definition~of~'#1'~will~not~be~altered.
1169 % }
1170 \msg_new:nnn { tensorstyles } { rank_indices_mismatch }
1171 { The~number~of~given~indices~and~rank~do~not~match~(#1) }
1172 \msg_new:nnn { tensorstyles } { index-type-option-not-define }
1173 { The~default~indices~is~only~accepting~contra~or~covariant~choice }
1174 \msg_new:nnn { tensorstyles } { index-style-option-not-defined }
1175 { Index~styles~(ie~contra~style~or~covariant~style)~on~accept~greek,~alph~or~arabic~choice
```

Declaring & Generate new group of commands

`\tensorssset{all}` First setup of the options for 'all' commands.  
*The following test files are used for this code: tensorstyles-test-005.*

```
1176 \@ifpackageloaded{mleftright}
1177 { \tensorssset{all}[scale-auto = mleftright] }
1178 { \tensorssset{all}[scale-auto = leftright] }
```

(End of definition for `\tensorssset{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.*

```
1179 \DeclareTensorsCmd{\tensor}[preset=full]
```

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

```
1180 \endinput
1181 </package>
```