

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

February 9, 2026

## Abstract

This document is the documented code of the LaTeX package `nicematrix`. It is *not* its user's guide. The guide of utilisation is the document `nicematrix.pdf` (with a French translation: `nicematrix-french.pdf`).

The development of the extension `nicematrix` is done on the following GitHub depot:  
<https://github.com/fpantigny/nicematrix>

## 1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registered for this package.

See: <http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf>  
<@=@=nicematrix>

First, we load `pgfcore` and the module `shapes`. We do so because it's not possible to use `\usepgfmodule` in `\ExplSyntaxOn`.

```
1 \RequirePackage{pgfcore}
2 \usepgfmodule{shapes}
```

We give the traditional declaration of a package written with the L3 programming layer.

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}

8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your~LaTeX~release~is~too~old. \\
11    You~need~at~least~the~version~of~2025-06-01. \\
12    If~you~use~Overleaf,~you~need~at~least~"TeXLive~2025".\\
13    The~package~'nicematrix'~won't~be~loaded.
14   }

15 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
16 \IfFormatAtLeastTF
17   { 2025-06-01 }
18   { }
19   { \msg_critical:nn { nicematrix } { latex-too-old } }
```

---

\*This document corresponds to the version 7.6 of `nicematrix`, at the date of 2026/02/01.

The command for the treatment of the options of `\usepackage` is at the end of this package for technical reasons.

```

20 \RequirePackage { amsmath }

21 \RequirePackage{array}[=2025/06/08] % v2.6j

22 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
23 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
24 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
25 \cs_generate_variant:Nn \@@_error:nn { n e }
26 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
27 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
28 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
29 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }

```

With Overleaf (and also in TeXPage), by default, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key H in order to have more information. That's why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key `messages-for-Overleaf` is used (at load-time).

```

30 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
31 {
32   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
33   { \msg_new:nnn { nicematrix } { #1 } { #2 \\\ #3 } }
34   {
35     \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 }

```

We keep also in memory in another message the complement of information (generally the list of the available keys) in order to write it the `log` file in all circumstances (it will be useful for the AI of some systems such as Prism).

```

36     \msg_new:nnn { nicematrix } { #1~+ } { #3 }
37   }
38 }

```

We also create a command which will generate usually an error but only a warning on Overleaf. The argument is given by curryfication.

```

39 \cs_new_protected:Npn \@@_error_or_warning:n
40 {
41   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
42   { \@@_warning:n }
43   { \@@_error:n }
44 }

```

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always “output”.

```

45 \bool_new:N \g_@@_messages_for_Overleaf_bool
46 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
47 {
48   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
49   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
50 }

51 \@@_msg_new:nn { mdwtab-loaded }
52 {
53   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
54   This~error~is~fatal.
55 }

56 \hook_gput_code:nnn { begindocument / end } { . }
57 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab-loaded } } }

```

## 2 Collecting options

The following technique allows to create user commands with the ability to put an arbitrary number of `[list of (key=val)]` after the name of the command.

*Example :*

`\@@_collect_options:n { \F } [x=a,y=b] [z=c,t=d] { arg }`

will be transformed in : `\F{x=a,y=b,z=c,t=d}{arg}`

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,

the command `\G` takes in an arbitrary number of optional arguments between square brackets.

Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

58 \cs_new_protected:Npn \@@_collect_options:n #1
59 {
60   \peek_meaning:NTF [
61     { \@@_collect_options:nw { #1 } }
62     { #1 { } }
63   }

```

We use `\NewDocumentCommand` in order to be able to allow nested brackets within the argument between `[` and `]`.

```

64 \NewDocumentCommand \@@_collect_options:nw { m r[] }
65 { \@@_collect_options:nn { #1 } { #2 } }
66
67 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
68 {
69   \peek_meaning:NTF [
70     { \@@_collect_options:nnw { #1 } { #2 } }
71     { #1 { #2 } }
72   }
73
74 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
75 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

## 3 Technical definitions

The following constants are defined only for efficiency in the tests.

```

76 \tl_const:Nn \c_@@_c_tl { c }
77 \tl_const:Nn \c_@@_l_tl { l }
78 \tl_const:Nn \c_@@_r_tl { r }
79 \tl_const:Nn \c_@@_all_tl { all }
80 \tl_const:Nn \c_@@_dot_tl { . }
81 \str_const:Nn \c_@@_r_str { r }
82 \str_const:Nn \c_@@_c_str { c }
83 \str_const:Nn \c_@@_l_str { l }

84 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
85 \tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }

```

The following token list will be used for definitions of user commands (with `\NewDocumentCommand`) with an embellishment using an *underscore* (there may be problems because of the catcode of the underscore).

```

86 \tl_new:N \l_@@_argspec_tl

```

```

87 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
88 \cs_generate_variant:Nn \str_set:Nn { N o }
89 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
90 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
91 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
92 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
93 \cs_generate_variant:Nn \dim_min:nn { v }
94 \cs_generate_variant:Nn \dim_max:nn { v }

95 \hook_gput_code:nnn { begindocument } { . }
96 {
97   \IfPackageLoadedTF { tikz }
98   {

```

In some constructions, we will have to use a `{pgfpicture}` which *must* be replaced by a `{tikzpicture}` if TikZ is loaded. However, this switch between `{pgfpicture}` and `{tikzpicture}` can't be done dynamically with a conditional because, when the TikZ library `external` is loaded by the user, the pair `\tikzpicture-\endtikzpicture` (or `\begin{tikzpicture}-\end{tikzpicture}`) must be statically “visible” (even when externalization is not activated).

That's why we create `\c_@@_pgfortikzpicture_tl` and `\c_@@_endpgfortikzpicture_tl` which will be used to construct in a `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

99   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
100   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
101 }
102 {
103   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
104   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
105 }
106 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2025, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

107 \IfClassLoadedTF { revtex4-1 }
108 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
109 {
110   \IfClassLoadedTF { revtex4-2 }
111   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112   {

```

Maybe one of the previous classes will be loaded inside another class... We try to detect that situation.

```

113   \cs_if_exist:NT \rvtx@ifformat@geq
114   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
115   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
116 }
117 }

```

If the final user uses `nicematrix`, PGF/TikZ will write instruction `\pgfsyspdfmark` in the `aux` file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the `aux` file. With the following code, we try to avoid that situation.

```

118 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
119 {
120   \iow_now:Nn \@mainaux
121   {
122     \ExplSyntaxOn
123     \cs_if_free:NT \pgfsyspdfmark
124     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
125     \ExplSyntaxOff
126   }
127   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
128 }

```

We define a command `\iddots` similar to `\ddots` (``) but with dots going forward (``). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package `mathdots`), we don't define it again.

```

129 \ProvideDocumentCommand \iddots { }
130 {
131   \mathinner
132   {
133     \mkern 1 mu
134     \box_move_up:nn { 1 pt } { \hbox { . } }
135     \mkern 2 mu
136     \box_move_up:nn { 4 pt } { \hbox { . } }
137     \mkern 2 mu
138     \box_move_up:nn { 7 pt }
139     { \vbox:n { \kern 7 pt \hbox { . } } }
140     \mkern 1 mu
141   }
142 }

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/TikZ nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

143 \hook_gput_code:nnn { begindocument } { . }
144 {
145   \IfPackageLoadedT { booktabs }
146   {
147     \iow_now:Nn \@mainaux
148     {
149       \ExplSyntaxOn
150       \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
151       \ExplSyntaxOff
152     }
153   }
154 }
155 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
156 {
157   \let @@_old_pgful@check@rerun \pgfutil@check@rerun

```

The new version of `\pgfutil@check@rerun` will not check the PGF nodes whose names start with `nm-` (which is the prefix for the nodes created by `nicematrix`).

```

158   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
159   {
160     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
161     { @@_old_pgful@check@rerun { ##1 } { ##2 } }
162   }
163 }

```

`\str_if_eq:ee`(TF) is slightly faster than `\str_if_eq:nn`(TF).

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`. The command `@@_everycr:` will be used only in `@@_some_initialization:`, itself in `\ar@ialign`.

```

164 \hook_gput_code:nnn { begindocument } { . }
165 {
166   \cs_set_protected:Npe @@_everycr:
167   {
168     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
169     { \noalign { @@_in_everycr: } }
170   }
171   \IfPackageLoadedTF { colortbl }
172   {

```

```

173 \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
174 \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
175 \cs_new_protected:Npn \@@_revert_colortbl:
176 {
177   \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
178   {
179     \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
180     \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
181   }
182 }

```

When `colortbl` is used, we have to catch the tokens `\columncolor` in the preamble because, otherwise, `colortbl` will catch them and the colored panels won't be drawn by `nicematrix` but by `colortbl` (with an output which is not perfect).

```

183 \cs_new_protected:Npn \@@_replace_columncolor:
184 {
185   \tl_replace_all:Nnn \g_@@_array_preamble_tl
186   { \columncolor }
187   { \@@_columncolor_preamble }

```

`\@@_column_preamble`, despite its name, will be defined with `\NewDocumentCommand` because it takes in an optional argument between square brackets in first position for the colorimetric space.

```

188   }
189 }
190 {
191   \cs_new_protected:Npn \@@_revert_colortbl: { }
192   \cs_new_protected:Npn \@@_replace_columncolor:
193   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded.

```

194 \def \CT@arc@ { }
195 \def \arrayrulecolor #1 # { \CT@arc { #1 } }
196 \def \CT@arc #1 #2
197 {
198   \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
199   { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
200 }

```

Idem for `\CT@drs@`.

```

201 \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
202 \def \CT@drs #1 #2
203 {
204   \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
205   { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
206 }
207 \def \hline
208 {
209   \noalign { \ifnum 0 = ` } \fi
210   \cs_set_eq:NN \hskip \vskip
211   \cs_set_eq:NN \vrule \hrule
212   \cs_set_eq:NN \@width \@height
213   { \CT@arc@ \vline }
214   \futurelet \reserved@a
215   \@xhline
216 }
217 }
218 }

```

We have to redefine `\cline` for several reasons. The command `\@@_cline:` will be linked to `\cline` in the beginning of `{NiceArrayWithDelims}`. The following commands must *not* be protected.

```

219 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
220 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
221 {

```

```

222 \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
223 \int_compare:nNnT { #1 } > { \c_one_int }
224 { \multispan { \int_eval:n { #1 - 1 } } & }
225 \multispan { \int_eval:n { #2 - #1 + 1 } }
226 {
227 \CT@arc@
228 \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`<sup>1</sup>

```

229 \skip_horizontal:N \c_zero_dim
230 }

```

Our `\everycr` has been modified. In particular, the creation of the `row` node is in the `\everycr` (maybe we should put it with the incrementation of `\c@iRow`). Since the following `\cr` correspond to a “false row”, we have to nullify `\everycr`.

```

231 \everycr { }
232 \cr
233 \noalign { \skip_vertical:n { - \arrayrulewidth } }
234 }

```

The following version of `\cline` spreads the array of a quantity equal to `\arrayrulewidth` as does `\hline`. It will be loaded excepted if the key `standard-cline` has been used.

```

235 \cs_set:Npn \@@_cline:

```

We have to act in a fully expandable way since there may be `\noalign` (in the `\multispan`) to detect. That’s why we use `\@@_cline_i:en`.

```

236 { \@@_cline_i:en { \l_@@_first_col_int } }

```

The command `\cline_i:nn` has two arguments. The first is the number of the current column (it *must* be used in that column). The second is a standard argument of `\cline` of the form *i-j* or the form *i*.

```

237 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
238 \cs_generate_variant:Nn \@@_cline_i:nn { e }
239 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
240 {
241 \tl_if_empty:nTF { #3 }
242 { \@@_cline_iii:w #1|#2-#2 \q_stop }
243 { \@@_cline_ii:w #1|#2-#3 \q_stop }
244 }
245 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
246 { \@@_cline_iii:w #1|#2-#3 \q_stop }
247 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
248 {

```

Now, `#1` is the number of the current column and we have to draw a line from the column `#2` to the column `#3` (both included).

```

249 \int_compare:nNnT { #1 } < { #2 }
250 { \multispan { \int_eval:n { #2 - #1 } } & }
251 \multispan { \int_eval:n { #3 - #2 + 1 } }
252 {
253 \CT@arc@
254 \leaders \hrule \@height \arrayrulewidth \hfill
255 \skip_horizontal:N \c_zero_dim
256 }

```

You look whether there is another `\cline` to draw (the final user may put several `\cline`).

```

257 \peek_meaning_remove_ignore_spaces:NTF \cline
258 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
259 { \everycr { } \cr }
260 }

```

---

<sup>1</sup>See question 99041 on TeX StackExchange.

The following command will be nullified in the environment {NiceTabular}, {NiceTabular\*} and {NiceTabularX}.

```

261 \cs_set:Nn \@@_math_toggle: { $ } % $

262 \cs_new_protected:Npn \@@_set_CTarc:n #1
263 {
264   \tl_if_blank:nF { #1 }
265   {
266     \tl_if_head_eq_meaning:nNTF { #1 } [
267       { \def \CT@arc@ { \color #1 } }
268       { \def \CT@arc@ { \color { #1 } } }
269     ]
270   }
271 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

272 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
273 {
274   \tl_if_head_eq_meaning:nNTF { #1 } [
275     { \def \CT@drsc@ { \color #1 } }
276     { \def \CT@drsc@ { \color { #1 } } }
277   ]

```

The following command must *not* be protected since it will be used to write instructions in the \g\_@@\_pre\_code\_before\_tl.

```

278 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
279 {
280   \tl_if_head_eq_meaning:nNTF { #2 } [
281     { #1 #2 }
282     { #1 { #2 } }
283   ]
284 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

The following command must be protected because of its use of the command \color.

```

285 \cs_new_protected:Npn \@@_color:n #1
286 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
287 \cs_generate_variant:Nn \@@_color:n { o }

288 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
289 {
290   \tl_set_rescan:Nno
291     #1
292     {
293       \char_set_catcode_other:N >
294       \char_set_catcode_other:N <
295     }
296   #1
297 }

```

The L3 programming layer provides scratch dimensions \l\_tmpa\_dim and \l\_tmpb\_dim. We create several more in the same spirit.

```

298 \dim_new:N \l_@@_tmpc_dim
299 \dim_new:N \l_@@_tmpd_dim

300 \tl_new:N \l_@@_tmpc_tl
301 \tl_new:N \l_@@_tmpd_tl

302 \int_new:N \l_@@_tmpc_int

```



## 4 Parameters

The following counter will count the environments `{NiceArray}`. The value of this counter will be used to prefix the names of the TikZ nodes created in the array.

```
303 \int_new:N \g_@@_env_int
```

The following command is only a syntactic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```
304 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in pgf nodes.

```
305 \NewExpandableDocumentCommand \NiceMatrixLastEnv { } { \int_use:N \g_@@_env_int }
```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```
306 \box_new:N \l_@@_the_array_box
```

The following command is only a syntactic shortcut. The `q` in `qpoint` means *quick*.

```
307 \cs_new_protected:Npn \@@_qpoint:n #1
308 { \pgfpointanchor { \@@_env: - #1 } { center } }
```

If the user uses `{NiceTabular}`, `{NiceTabular*}` or `{NiceTabularX}`, we will raise the following flag.

```
309 \bool_new:N \l_@@_tabular_bool
```

`\g_@@_delims_bool` will be true for the environments with delimiters (ex. : `{pNiceMatrix}`, `{pNiceArray}`, `\pAutoNiceMatrix`, etc.).

```
310 \bool_new:N \g_@@_delims_bool
311 \bool_gset_true:N \g_@@_delims_bool
```

In fact, if there is delimiters in the preamble of `{NiceArray}` (eg: `[cccc]`), this boolean will be set to false.

The following boolean will be equal to `true` in the environments which have a preamble (provided by the final user): `{NiceTabular}`, `{NiceArray}`, `{pNiceArray}`, etc.

```
312 \bool_new:N \l_@@_preamble_bool
313 \bool_set_true:N \l_@@_preamble_bool
```

We need a special treatment for `{NiceMatrix}` when `vlines` is not used, in order to retrieve `\arraycolsep` on both sides.

```
314 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool
```

The following counter will count the environments `{NiceMatrixBlock}`.

```
315 \int_new:N \g_@@_NiceMatrixBlock_int
```

It's possible to put tabular notes (with `\tabularnote`) in the caption if that caption is composed *above* the tabular. In such case, we will count in `\g_@@_notes_caption_int` the number of uses of the command `\tabularnote` *without optional argument* in that caption.

```
316 \int_new:N \g_@@_notes_caption_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width (but, if the key `columns-width` is used with the special value `auto`, the boolean `\l_@@_auto_columns_width_bool` also will be raised).

```
317 \dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{...}`, `m{...}`, `b{...}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
318 \dim_new:N \l_@@_col_width_dim
319 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

```
320 \int_new:N \g_@@_row_total_int
321 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node`: to avoid to create the same row-node twice (at the end of the array).

```
322 \int_new:N \g_@@_last_row_node_int
```

The following counter corresponds to the key `nb-rows` of the command `\RowStyle`.

```
323 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c` and `j`. For example, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

```
324 \tl_new:N \l_@@_hpos_cell_tl
325 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

When there is a mono-column block (created by the command `\Block`), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the `\g_@@_blocks_wd_dim` and, after the construction of the box `\l_@@_cell_box`, we change the width of that box to take into account the length `\g_@@_blocks_wd_dim`.

```
326 \dim_new:N \g_@@_blocks_wd_dim
```

Idem for the mono-row blocks.

```
327 \dim_new:N \g_@@_blocks_ht_dim
328 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension correspond to the key `width` (which may be fixed in `\NiceMatrixOptions` but also in an environment `\NiceTabular`).

```
329 \dim_new:N \l_@@_width_dim
```

The clist `\g_@@_names_clist` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
330 \clist_new:N \g_@@_names_clist
```

We want to know whether we are in an environment of `nicematrix` because we will raise an error if the user tries to use nested environments.

```
331 \bool_new:N \l_@@_in_env_bool
```

The following key corresponds to the key `notes/detect_duplicates`.

```
332 \bool_new:N \l_@@_notes_detect_duplicates_bool
333 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```

334 \bool_new:N \l_@@_initial_open_bool
335 \bool_new:N \l_@@_final_open_bool
336 \bool_new:N \l_@@_Vbrace_bool

```

If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

```

337 \dim_new:N \l_@@_tabular_width_dim

```

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```

338 \dim_new:N \l_@@_rule_width_dim

```

The key `color` in a command of rule such as `\Hline` (or the specifier “|” in the preamble of an environment).

```

339 \tl_new:N \l_@@_rule_color_tl

```

The following boolean will be raised when the command `\rotate` is used.

```

340 \bool_new:N \g_@@_rotate_bool

```

The following boolean will be raised when the command `\rotate` is used with the key `c`.

```

341 \bool_new:N \g_@@_rotate_c_bool

```

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag (the `X` columns of `nicematrix` are inspired by those of `tabularx`). You will use that flag for the blocks.

```

342 \bool_new:N \l_@@_X_bool

```

`\l_@@_V_of_X_bool` during the construction of the preamble when a column of type `X` uses the key `V` (whose name is inspired by the columns `V` of the extension `varwidth`).

```

343 \bool_new:N \l_@@_V_of_X_bool

```

The flag `\g_@@_V_of_X_bool` will be raised when there is at least in the tabular a column of type `X` using the key `V`.

```

344 \bool_new:N \g_@@_V_of_X_bool

```

```

345 \bool_new:N \g_@@_caption_finished_bool

```

The following boolean will be raised when the key `no-cell-nodes` is used.

```

346 \bool_new:N \l_@@_no_cell_nodes_bool

```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }`).

```

347 \tl_new:N \g_@@_aux_tl

```

During the second run, if information concerning the current environment has been found in the `aux` file, the following flag will be raised. It will be used, for instance to disable several constructions (continuous dotted lines, and colored backgrounds) during the first compilation (in order to speed up it).

```

348 \bool_new:N \g_@@_aux_found_bool

```

In particular, in that `aux` file, there will be, for each environment of `nicematrix`, an affectation for the the following sequence that will contain information about the size of the array.

```

349 \seq_new:N \g_@@_size_seq

```

```

350 \tl_new:N \g_@@_left_delim_tl

```

```

351 \tl_new:N \g_@@_right_delim_tl

```

The token list `\g_@@_user_preamble_tl` will contain the preamble provided by the the final user of `nicematrix` (eg the preamble of an environment `{NiceTabular}`).

```
352 \tl_new:N \g_@@_user_preamble_tl
```

The token list `\g_@@_array_preamble_tl` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of `array`).

```
353 \tl_new:N \g_@@_array_preamble_tl
```

For `\multicolumn`.

```
354 \tl_new:N \g_@@_preamble_tl
```

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`.

```
355 \tl_new:N \l_@@_columns_type_tl
```

```
356 \str_set:Nn \l_@@_columns_type_tl { c }
```

The following parameters correspond to the keys `down`, `up` and `middle` of a command such as `\Cdots`. Usually, the final user doesn't use that keys directly because he uses the syntax with the embellishments `_`, `^` and `..`.

```
357 \tl_new:N \l_@@_xdots_down_tl
```

```
358 \tl_new:N \l_@@_xdots_up_tl
```

```
359 \tl_new:N \l_@@_xdots_middle_tl
```

We will store in the following sequence information provided by the instructions `\rowlistcolors` in the main array (not in the `\CodeBefore`).

```
360 \seq_new:N \g_@@_rowlistcolors_seq
```

```
361 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
362 {
```

```
363   \if_mode_math: \else:
```

```
364     \@@_fatal:n { Outside~math~mode }
```

```
365   \fi:
```

```
366 }
```

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analysis of the preamble of the array.

```
367 \seq_new:N \g_@@_cols_vlism_seq
```

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

```
368 \colorlet { nicematrix-last-col } { . }
```

```
369 \colorlet { nicematrix-last-row } { . }
```

The following string is the name of the current environment or the current command of `nicematrix` (despite its name which contains `env`).

```
370 \str_new:N \g_@@_name_env_str
```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

```
371 \str_new:N \g_@@_com_or_env_str
```

```
372 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

```
373 \bool_new:N \l_@@_bold_row_style_bool
```

```
374 \clist_new:N \g_@@_cbic_clist
```

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains *env*). This command must *not* be protected since it will be used in error messages and we have to use `\str_if_eq:eeTF` and not `\tl_if_eq:eeTF` because we need to be fully expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

375 \cs_new:Npn \@@_full_name_env:
376 {
377   \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
378   { command \space \c_backslash_str \g_@@_name_env_str }
379   { environment \space \{ \g_@@_name_env_str \} }
380 }

381 \tl_new:N \g_@@_cell_after_hook_tl

```

For the key code of the command `\SubMatrix` (itself in the main `\CodeAfter`), we will use the following token list.

```

382 \tl_new:N \l_@@_code_tl

```

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form *i-j*) will be created.

```

383 \tl_new:N \l_@@_pgf_node_code_tl

```

The so-called `\CodeBefore` is split in two parts because we want to control the order of execution of some instructions.

```

384 \tl_new:N \g_@@_pre_code_before_tl
385 \tl_new:N \g_nicematrix_code_before_tl

```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

The so-called `\CodeAfter` is split in two parts because we want to control the order of execution of some instructions.

```

386 \tl_new:N \g_@@_pre_code_after_tl
387 \tl_new:N \g_nicematrix_code_after_tl

```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

```

388 \bool_new:N \l_@@_in_code_after_bool

```

The following parameter will be raised when a block contains an ampersand (&) in its content (=label).

```

389 \bool_new:N \l_@@_ampersand_bool

```

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```

390 \int_new:N \l_@@_old_iRow_int
391 \int_new:N \l_@@_old_jCol_int

```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

```

392 \seq_new:N \l_@@_custom_line_commands_seq

```

The following token list corresponds to the key `rules/color` available in the environments.

```

393 \tl_new:N \l_@@_rules_color_tl

```

The sum of the weights of all the X-columns in the preamble.

```
394 \fp_new:N \g_@@_total_X_weight_fp
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `l_@@_x_columns_dim` will be the width of X-columns of weight 1.0 (the width of a column of weight  $x$  will be that dimension multiplied by  $x$ ). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
395 \bool_new:N \l_@@_X_columns_aux_bool
396 \dim_new:N \l_@@_X_columns_dim
```

```
397 \dim_new:N \l_@@_brace_shift_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
398 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```
399 \bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the TikZ nodes are constructed only in the non empty cells).

```
400 \bool_new:N \g_@@_not_empty_cell_bool
```

```
401 \tl_new:N \l_@@_code_before_tl
```

```
402 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

```
403 \tl_new:N \g_@@_row_style_tl
```

The following dimensions will be used when drawing the dotted lines.

```
404 \dim_new:N \l_@@_x_initial_dim
```

```
405 \dim_new:N \l_@@_y_initial_dim
```

```
406 \dim_new:N \l_@@_x_final_dim
```

```
407 \dim_new:N \l_@@_y_final_dim
```

```
408 \dim_new:N \g_@@_dp_row_zero_dim
```

```
409 \dim_new:N \g_@@_ht_row_zero_dim
```

```
410 \dim_new:N \g_@@_ht_row_one_dim
```

```
411 \dim_new:N \g_@@_dp_ante_last_row_dim
```

```
412 \dim_new:N \g_@@_ht_last_row_dim
```

```
413 \dim_new:N \g_@@_dp_last_row_dim
```

Some cells will be declared as “empty” (for example a cell with an instruction `\Cdots`).

```
414 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

```
415 \dim_new:N \g_@@_width_last_col_dim
```

```
416 \dim_new:N \g_@@_width_first_col_dim
```

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

`{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```
417 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{ name}`. A block with the key `hvlines` won't appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
418 \seq_new:N \g_@@_pos_of_blocks_seq
```

In fact, this sequence will also contain the positions of the cells with a `\diagbox`. The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules (which respect the blocks).

In the `\CodeBefore`, the value of `\g_@@_pos_of_blocks_seq` will be the value read in the `aux` file from a previous run. However, in the `\CodeBefore`, the commands `\EmptyColumn` and `\EmptyRow` will write virtual positions of blocks in the following sequence.

```
419 \seq_new:N \g_@@_future_pos_of_blocks_seq
```

They will be added to `\g_@@_pos_of_blocks_seq` after the computation of the “empty corners”.

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by `\Cdots`, `\Vdots`, `\Ddots`, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: `{imin}{jmin}{imax}{jmax}{ name}`.

```
420 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That's why we keep the information of all that stroken blocks in the following sequence.

```
421 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
422 \clist_new:N \l_@@_corners_cells_clist
```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

```
423 \seq_new:N \g_@@_submatrix_names_seq
```

The following flag will be raised if the key `width` is used in an environment `{NiceTabular}` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```
424 \bool_new:N \l_@@_width_used_bool
```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with  $n > 1$  is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of  $n$ ) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
425 \seq_new:N \g_@@_multicolumn_cells_seq
```

```
426 \seq_new:N \g_@@_multicolumn_sizes_seq
```

By default, the diagonal lines will be parallelized<sup>2</sup>. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```
427 \int_new:N \g_@@_ddots_int
428 \int_new:N \g_@@_iddots_int
```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  and  $\Delta_y$  of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the  $\Delta_x$  and  $\Delta_y$  of the first `\Iddots` diagonal.

```
429 \dim_new:N \g_@@_delta_x_one_dim
430 \dim_new:N \g_@@_delta_y_one_dim
431 \dim_new:N \g_@@_delta_x_two_dim
432 \dim_new:N \g_@@_delta_y_two_dim
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

```
433 \int_new:N \l_@@_row_min_int
434 \int_new:N \l_@@_row_max_int
435 \int_new:N \l_@@_col_min_int
436 \int_new:N \l_@@_col_max_int

437 \int_new:N \l_@@_initial_i_int
438 \int_new:N \l_@@_initial_j_int
439 \int_new:N \l_@@_final_i_int
440 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
441 \int_new:N \l_@@_start_int
442 \int_set_eq:NN \l_@@_start_int \c_one_int
443 \int_new:N \l_@@_end_int
444 \int_new:N \l_@@_local_start_int
445 \int_new:N \l_@@_local_end_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. Each sub-matrix is represented by an “object” of the form `{i}{j}{k}{l}` where  $i$  and  $j$  are the number of row and column of the upper-left cell and  $k$  and  $l$  the number of row and column of the lower-right cell.

```
446 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

```
447 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `opacity`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

```
448 \tl_new:N \l_@@_fill_tl
449 \tl_new:N \l_@@_opacity_tl
450 \tl_new:N \l_@@_draw_tl
451 \seq_new:N \l_@@_tikz_seq
452 \clist_new:N \l_@@_borders_clist
453 \dim_new:N \l_@@_rounded_corners_dim
```

---

<sup>2</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.



The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```
454 \dim_new:N \l_@@_tab_rounded_corners_dim
```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

```
455 \tl_new:N \l_@@_color_tl
```

In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of `tikz` keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

```
456 \dim_new:N \l_@@_offset_dim
```

Here is the dimension for the width of the rule when a block (created by `\Block`) is stroked or when the key `hvlines` is used.

```
457 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
458 \str_new:N \l_@@_hpos_block_str
459 \str_set:Nn \l_@@_hpos_block_str { c }
460 \bool_new:N \l_@@_hpos_of_block_cap_bool
461 \bool_new:N \l_@@_p_block_bool
```

If the final user has used the special color “`nocolor`”, the following flag will be raised.

```
462 \bool_new:N \l_@@_nocolor_used_bool
```

For the vertical position, the possible values are `c`, `t`, `b`, `T` and `B` (but `\l_@@_vpos_block_str` will remain empty if the user doesn’t use a key for the vertical position).

```
463 \str_new:N \l_@@_vpos_block_str
```

Used when the key `draw-first` is used for `\Ddots` or `\Iddots`.

```
464 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
465 \bool_new:N \l_@@_vlines_block_bool
466 \bool_new:N \l_@@_hlines_block_bool
```

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

```
467 \int_new:N \g_@@_block_box_int

468 \dim_new:N \l_@@_submatrix_extra_height_dim
469 \dim_new:N \l_@@_submatrix_left_xshift_dim
470 \dim_new:N \l_@@_submatrix_right_xshift_dim
471 \clist_new:N \l_@@_hlines_clist
472 \clist_new:N \l_@@_vlines_clist
473 \clist_new:N \l_@@_submatrix_hlines_clist
474 \clist_new:N \l_@@_submatrix_vlines_clist
```

The following key is set when the keys `hvlines` and `hvlines-except-borders` are used. It’s used only to change slightly the clipping path set by the key `rounded-corners` (for a `{tabular}`).

```
475 \bool_new:N \l_@@_hvlines_bool
```

The following flag will be used by (for instance) `\@@_vline_ii:`. When `\l_@@_dotted_bool` is true, a dotted line (with our system) will be drawn.

```
476 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to true during the composition of a caption specified (by the key `caption`).

```
477 \bool_new:N \l_@@_in_caption_bool
```

## Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

### • First row

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
478 \int_new:N \l_@@_first_row_int
479 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

### • First column

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```
480 \int_new:N \l_@@_first_col_int
481 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

### • Last row

The counter `\l_@@_last_row_int` is the number of the potential “last row”, as specified by the key `last-row`. A value of `-2` means that there is no “last row”. A value of `-1` means that there is a “last row” but we don’t know the number of that row (the key `last-row` has been used without value and the actual value has not still been read in the `aux` file).

```
482 \int_new:N \l_@@_last_row_int
483 \int_set:Nn \l_@@_last_row_int { -2 }
```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.<sup>3</sup>

```
484 \bool_new:N \l_@@_last_row_without_value_bool
```

Idem for `\l_@@_last_col_without_value_bool`

```
485 \bool_new:N \l_@@_last_col_without_value_bool
```

---

<sup>3</sup>We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

- **Last column**

For the potential “last column”, we use an integer. A value of  $-2$  means that there is no last column. A value of  $-1$  means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of  $0$  means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to  $0$ .

```
486 \int_new:N \l_@@_last_col_int
487 \int_set:Nn \l_@@_last_col_int { -2 }
```

However, we have also a boolean. Consider the following code:

```
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}
```

In such a code, the “last column” specified by the key `last-col` is not used. We want to be able to detect such a situation and we create a boolean for that job.

```
488 \bool_new:N \g_@@_last_col_found_bool
```

This boolean is set to `false` at the end of `\@@_pre_array_after_CodeBefore:.`

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

```
489 \bool_new:N \l_@@_in_last_col_bool
```

## Some utilities

```
490 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
491 {
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
492 \def \l_tmpa_tl { #1 }
493 \def \l_tmpb_tl { #2 }
494 }
```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers. The second argument is `\c@iRow` or `\c@jCol`.

```
495 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
496 {
497   \clist_if_in:NnF #1 { all }
498   {
499     \clist_clear:N \l_tmpa_clist
500     \clist_map_inline:Nn #1
501     {
502       \tl_if_head_eq_meaning:nNTF { ##1 } -
503       {
```

If we have yet the number of columns or the number of columns (because they have been computed during a previous run and written on the `aux` file), we can compute the actual position of the rule with a negative position.

```
504       \int_if_zero:nF { #2 }
505       {
506         \clist_put_right:Ne \l_tmpa_clist
507         { \int_eval:n { #2 + (##1) + 1 } }
508       }
509     }
510   }
```

We recall than `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```

511         \tl_if_in:nnTF { ##1 } { - }
512         { \@@_cut_on_hyphen:w ##1 \q_stop }
513         {

```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```

514         \def \l_tmpa_tl { ##1 }
515         \def \l_tmpb_tl { ##1 }
516     }
517     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
518     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
519 }
520 }
521 \tl_set_eq:NN #1 \l_tmpa_clist
522 }
523 }

```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- when the special character “:” is used in order to put the label of a so-called “dotted line” on the line, a margin of `\c_@@_innersep_middle_dim` will be added around the label.

```

524 \hook_gput_code:nnn { begindocument } { . }
525 {
526     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
527     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
528 }

```

## 5 The command `\tabularnote`

Of course, it’s possible to use `\tabularnote` in the main tabular. But there is also the possibility to use that command in the caption of the tabular. And the caption may be specified by two means:

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It’s also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:
  - The number of tabular notes present in the caption will be written on the aux file and available in `\g_@@_notes_caption_int`.<sup>4</sup>
  - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\NoValue`).

---

<sup>4</sup>More precisely, it’s the number of tabular notes which do not use the optional argument of `\tabularnote`.

- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

The LaTeX counter `tabularnote` will be used to count the tabular notes during the construction of the array (this counter won't be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```
529 \newcounter { tabularnote }
```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That's why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

```
530 \int_new:N \g_@@_tabularnote_int
531 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
532 \seq_new:N \g_@@_notes_seq
533 \seq_new:N \g_@@_notes_in_caption_seq
```

Before the actual tabular notes, it's possible to put a text specified by the key `tabularnote` of the environment. The token list `\g_@@_tabularnote_tl` corresponds to the value of that key.

```
534 \tl_new:N \g_@@_tabularnote_tl
```

We prepare the tools for the formatting of the references of the footnotes (in the tabular itself). There may have several references of footnote at the same point and we have to take into account that point.

```
535 \seq_new:N \l_@@_notes_labels_seq
536 \newcounter { nicematrix_draft }
537 \cs_new_protected:Npn \@@_notes_format:n #1
538 {
539   \setcounter { nicematrix_draft } { #1 }
540   \@@_notes_style:n { nicematrix_draft }
541 }
```

The following function can be redefined by using the key `notes/style`.

```
542 \cs_new:Npn \@@_notes_style:n #1 { \textit { \alph { #1 } } }
```

The following function can be redefined by using the key `notes/label-in-tabular`.

```
543 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
```

The following function can be redefined by using the key `notes/label-in-list`.

```
544 \cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript { #1 } }
```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a tabular which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```
545 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

The tabular notes will be available for the final user only when `enumitem` is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by `enumitem` (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether `enumitem` has been loaded only at the beginning of the document (we want to allow the user to load `enumitem` after `nicematrix`).

```
546 \hook_gput_code:nnn { begindocument } { . }
547 {
548   \IfPackageLoadedTF { enumitem }
549   {
```

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

550     \newlist { tabularnotes } { enumerate } { 1 }
551     \setlist [ tabularnotes ]
552     {
553         topsep = \c_zero_dim ,
554         noitemsep ,
555         leftmargin = * ,
556         align = left ,
557         labelsep = \c_zero_dim ,
558         label =
559             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
560     }
561     \newlist { tabularnotes* } { enumerate* } { 1 }
562     \setlist [ tabularnotes* ]
563     {
564         afterlabel = \nobreak ,
565         itemjoin = \quad ,
566         label =
567             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
568     }

```

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

```

569     \NewDocumentCommand { \tabularnote } { o m }
570     {
571         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
572         {
573             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
574             { \@@_error:n { tabularnote~forbidden } }
575             {

```

The second argument of `\@@_tabularnote_caption:nn` ou `\@@_tabularnote:nn` is provided by cur-ryfication.

```

576         \bool_if:NTF \l_@@_in_caption_bool
577         {
578             \tl_if_novalue:nTF { #1 }
579             { \@@_tabularnote_caption:nn { #1 } }
580             { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
581         }
582         {
583             \tl_if_novalue:nTF { #1 }
584             { \@@_tabularnote:nn { #1 } }
585             { \@@_tabularnote:nn { \exp_not:n { #1 } } }
586         }
587         { #2 }
588     }
589 }
590 }
591 }
592 {
593     \NewDocumentCommand \tabularnote { o m }
594     { \@@_err_enumitem_not_loaded: }
595 }
596 }
597 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
598 {
599     \@@_error_or_warning:n { enumitem~not~loaded }
600     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
601 }

```

```

602 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
603 { \tl_if_novalue:nT { #1 } { #3 } }

```

For the version in normal conditions, that is to say not in the `caption`. `#1` is the optional argument of `\tabularnote` (maybe equal to the special marker expressed by `\NoValue`) and `#2` is the mandatory argument of `\tabularnote`.

```

604 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
605 {

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in `\g_@@_notes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

606   \int_zero:N \l_tmpa_int
607   \bool_if:NT \l_@@_notes_detect_duplicates_bool
608   {

```

We recall that each component of `\g_@@_notes_seq` is a kind of couple of the form

`{label}{text of the tabularnote}`.

If the user have used `\tabularnote` without the optional argument, the `label` will be the special marker expressed by `\NoValue`.

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabularnote`.

```

609   \int_zero:N \l_tmpb_int
610   \seq_map_indexed_inline:Nn \g_@@_notes_seq
611   {
612     \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
613     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
614     {
615       \tl_if_novalue:nTF { #1 }
616       { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
617       { \int_set:Nn \l_tmpa_int { ##1 } }
618       \seq_map_break:
619     }
620   }
621   \int_if_zero:nF { \l_tmpa_int }
622   { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
623 }
624 \int_if_zero:nT { \l_tmpa_int }
625 {
626   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
627   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
628 }
629 \seq_put_right:Ne \l_@@_notes_labels_seq
630 {
631   \tl_if_novalue:nTF { #1 }
632   {
633     \@@_notes_format:n
634     {
635       \int_eval:n
636       {
637         \int_if_zero:nTF { \l_tmpa_int }
638         { \c@tabularnote }
639         { \l_tmpa_int }
640       }
641     }
642   }
643   { #1 }
644 }
645 \peek_meaning:NF \tabularnote
646 {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```
647 \hbox_set:Nn \l_tmpa_box
648 {
```

We remind that it is the command `\@@_notes_label_in_tabular:n` that will put the labels in a `\textsuperscript`.

```
649 \@@_notes_label_in_tabular:n
650 { \seq_use:Nn \l_@@_notes_labels_seq { , } }
651 }
```

We want the (last) tabular note referenceable (with the standard command `\label`).

```
652 \int_gdecr:N \c@tabularnote
653 \int_set_eq:NN \l_tmpa_int \c@tabularnote
```

The following line is only to avoid error messages for multiply defined labels when the package `hyperref` is used.

```
654 \int_gincr:N \g_@@_tabularnote_int
655 \refstepcounter { tabularnote }
656 \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
657 { \int_gincr:N \c@tabularnote }
658 \seq_clear:N \l_@@_notes_labels_seq
659 \bool_lazy_or:nnTF
660 { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
661 { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
662 {
663 \hbox_overlap_right:n { \box_use:N \l_tmpa_box }
```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array?`) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```
664 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
665 }
666 { \box_use:N \l_tmpa_box }
667 }
668 }
```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

```
669 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
670 {
671 \bool_if:NTF \g_@@_caption_finished_bool
672 {
673 \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
674 { \int_gzero:N \c@tabularnote }
```

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

```
675 \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
676 { \@@_error:n { Identical~notes~in~caption } }
677 }
678 {
```

In the following code, we are in the first composition of the caption or at the first `\tabularnote` of the second composition.

```
679 \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
680 {
```



Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

```

681         \bool_gset_true:N \g_@@_caption_finished_bool
682         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
683         \int_gzero:N \c@tabularnote
684     }
685     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
686 }

```

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

```

687     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
688     \seq_put_right:Ne \l_@@_notes_labels_seq
689     {
690         \tl_if_novalue:nTF { #1 }
691         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
692         { #1 }
693     }
694     \peek_meaning:NF \tabularnote
695     {
696         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
697         \seq_clear:N \l_@@_notes_labels_seq
698     }
699 }

700 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
701 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

## 6 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

**#1** is the name of the node which will be created; **#2** and **#3** are the coordinates of one of the corner of the rectangle; **#4** and **#5** are the coordinates of the opposite corner.

```

702 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
703 {
704     \begin { pgfscope }
705     \pgfset
706     {
707         inner~sep = \c_zero_dim ,
708         minimum~size = \c_zero_dim
709     }
710     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
711     \pgfnode
712     { rectangle }
713     { center }
714     {
715         \vbox_to_ht:nn
716         { \dim_abs:n { #5 - #3 } }
717         {
718             \vfill
719             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
720         }
721     }
722     { #1 }
723     { }
724     \end { pgfscope }
725 }

```

The command `\@@pgf_rect_node:nnn` is a variant of `\@@pgf_rect_node:nnnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

726 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
727 {
728   \begin { pgfscope }
729   \pgfset
730   {
731     inner~sep = \c_zero_dim ,
732     minimum~size = \c_zero_dim
733   }
734   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
735   \pgfpointdiff { #3 } { #2 }
736   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
737   \pgfnode
738   { rectangle }
739   { center }
740   {
741     \vbox_to_ht:nn
742     { \dim_abs:n \l_tmpb_dim }
743     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
744   }
745   { #1 }
746   { }
747   \end { pgfscope }
748 }

```

## 7 The options

The following parameter corresponds to the keys `caption`, `short-caption` and `label` of the environment `{NiceTabular}`.

```

749 \tl_new:N \l_@@_caption_tl
750 \tl_new:N \l_@@_short_caption_tl
751 \tl_new:N \l_@@_label_tl

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

752 \bool_new:N \l_@@_caption_above_bool

```

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

753 \bool_new:N \l_@@_standard_cline_bool

```

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

```

754 \dim_new:N \l_@@_cell_space_top_limit_dim
755 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

The following parameter corresponds to the key `xdots/horizontal_labels`.

```

756 \bool_new:N \l_@@_xdots_h_labels_bool

```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```
757 \dim_new:N \l_@@_xdots_inter_dim
758 \hook_gput_code:nnn { begindocument } { . }
759 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }
```

The unit is em and that's why we fix the dimension after the preamble.

The following dimension is the distance between a node (in fact an anchor of that node) and a dotted line (for real dotted lines, the actual distance may, of course, be a bit larger, depending of the exact position of the dots).

```
760 \dim_new:N \l_@@_xdots_shorten_start_dim
761 \dim_new:N \l_@@_xdots_shorten_end_dim
762 \hook_gput_code:nnn { begindocument } { . }
763 {
764   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
765   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
766 }
```

The unit is em and that's why we fix the dimension after the preamble.

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```
767 \dim_new:N \l_@@_xdots_radius_dim
768 \hook_gput_code:nnn { begindocument } { . }
769 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

The unit is em and that's why we fix the dimension after the preamble.

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

```
770 \tl_new:N \l_@@_xdots_line_style_tl
771 \tl_const:Nn \c_@@_standard_tl { standard }
772 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

```
773 \bool_new:N \l_@@_light_syntax_bool
774 \bool_new:N \l_@@_light_syntax_expanded_bool
```

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain **an integer** (which represents the number of the row to which align the array).

```
775 \tl_new:N \l_@@_baseline_tl
776 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```
777 \bool_new:N \l_@@_amp_in_blocks_bool
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

```
778 \bool_new:N \l_@@_exterior_arraycolsep_bool
```

The flag `\l_@@_parallelize_diags_bool` controls whether the diagonals are parallelized. The initial value is `true`.

```
779 \bool_new:N \l_@@_parallelize_diags_bool
780 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be within NW, SW, NE and SE.

```
781 \clist_new:N \l_@@_corners_clist
```

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
782 \bool_new:N \l_@@_nullify_dots_bool
```

When the key `respect-arraystretch` is used, the following command will be nullified.

```
783 \cs_new_protected:Npn \@@_reset_arraystretch: { \def \arraystretch { 1 } }
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

```
784 \bool_new:N \l_@@_auto_columns_width_bool
```

The following boolean corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
785 \bool_new:N \g_@@_create_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the TikZ nodes created in the array from outside the environment.

```
786 \str_new:N \l_@@_name_str
```

The boolean `\l_@@_medium_nodes_bool` will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

```
787 \bool_new:N \l_@@_medium_nodes_bool
```

```
788 \bool_new:N \l_@@_large_nodes_bool
```

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

```
789 \bool_new:N \l_@@_except_borders_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
790 \dim_new:N \l_@@_left_margin_dim
```

```
791 \dim_new:N \l_@@_right_margin_dim
```

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

```
792 \dim_new:N \l_@@_extra_left_margin_dim
```

```
793 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
794 \tl_new:N \l_@@_end_of_row_tl
```

```
795 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

```
796 \tl_new:N \l_@@_xdots_color_tl
```

The following token list corresponds to the key `delimiters/color`.

```
797 \tl_new:N \l_@@_delimiters_color_tl
```

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is function of its size. That's why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

```
798 \bool_new:N \l_@@_delimiters_max_width_bool
```

```
799 \keys_define:nn { nicematrix / xdots }
800 {
801   nullify .bool_set:N = \l_@@_nullify_dots_bool ,
802   nullify .default:n = true ,
803   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
804   brace-shift .value_required:n = true ,
805   brace-shift+ .code:n =
806     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
807   brace-shift+ .value_required:n = true ,
808   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
809   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
810   shorten-start .code:n =
811     \hook_gput_code:nnn { begindocument } { . }
812     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
813   shorten-start .value_required:n = true ,
814   shorten-start+ .code:n =
815     \hook_gput_code:nnn { begindocument } { . }
816     { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
817   shorten-start+ .meta:n = { shorten-start += #1 } ,
818   shorten-start+ .value_required:n = true ,
819   shorten-end .code:n =
820     \hook_gput_code:nnn { begindocument } { . }
821     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
822   shorten-end .value_required:n = true ,
823   shorten-end+ .code:n =
824     \hook_gput_code:nnn { begindocument } { . }
825     { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
826   shorten-end+ .value_required:n = true ,
827   shorten-end~+ .meta:n = { shorten-end += #1 } ,
828   shorten .code:n =
829     \hook_gput_code:nnn { begindocument } { . }
830     {
831       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
832       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
833     } ,
834   shorten .value_required:n = true ,
835   shorten+ .code:n =
836     \hook_gput_code:nnn { begindocument } { . }
837     {
838       \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
839       \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
840     } ,
841   shorten~+ .meta:n = { shorten+ = #1 } ,
842   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
843   horizontal-labels .default:n = true ,
844   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
845   horizontal-label .default:n = true ,
846   line-style .code:n =
847   {
```

```

848     \bool_lazy_or:nnTF
849     { \cs_if_exist_p:N \tikzpicture }
850     { \str_if_eq_p:nn { #1 } { standard } }
851     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
852     { \@@_error:n { bad-option-for-line-style } }
853   } ,
854   line-style .value_required:n = true ,
855   color .tl_set:N = \l_@@_xdots_color_tl ,
856   color .value_required:n = true ,
857   radius .code:n =
858     \hook_gput_code:nnn { begindocument } { . }
859     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
860   radius .value_required:n = true ,
861   inter .code:n =
862     \hook_gput_code:nnn { begindocument } { . }
863     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
864   radius .value_required:n = true ,

```

The options `down`, `up` and `middle` are not documented for the final user because he should use the syntax with `^`, `_` and `:`. We use `\tl_put_right:Nn` and not `\tl_set:Nn` (or `.tl_set:N`) because we don't want a direct use of `up=...` erased by an absent `^{\dots}`.

```

865   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
866   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
867   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, will be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

868   draw-first .code:n = \prg_do_nothing: ,
869   unknown .code:n =
870     \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
871 }

```

```

872 \keys_define:nn { nicematrix / rules }
873 {
874   color .tl_set:N = \l_@@_rules_color_tl ,
875   color .value_required:n = true ,
876   width .dim_set:N = \arrayrulewidth ,
877   width .value_required:n = true ,
878   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
879 }

```

First, we define a set of keys “`nicematrix / Global`” which will be used (with the mechanism of `.inherit:n`) by other sets of keys.

```

880 \keys_define:nn { nicematrix / Global }
881 {
882   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
883   brace-shift .value_required:n = true ,
884   brace-shift+ .code:n =
885     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
886   brace-shift+ .value_required:n = true ,
887   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
888   caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
889   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
890   color-inside .code:n = \@@_fatal:n { key-color-inside } ,
891   colortbl-like .code:n = \@@_fatal:n { key-color-inside } ,
892   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
893   ampersand-in-blocks .default:n = true ,
894   &-in-blocks .meta:n = ampersand-in-blocks ,
895   no-cell-nodes .code:n =
896     \bool_set_true:N \l_@@_no_cell_nodes_bool
897     \cs_set_protected:Npn \@@_node_cell:

```

```

898     { \set@color \box_use_drop:N \l_@@_cell_box } ,
899 no-cell-nodes .value_forbidden:n = true ,
900 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
901 rounded-corners .default:n = 4 pt ,
902 custom-line .code:n = \@@_custom_line:n { #1 } ,
903 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
904 rules .value_required:n = true ,
905 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
906 standard-cline .default:n = true ,
907 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
908 cell-space-top-limit .value_required:n = true ,
909 cell-space-top-limit+ .code:n =
910     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
911 cell-space-top-limit+ .value_required:n = true ,
912 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
913 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
914 cell-space-bottom-limit .value_required:n = true ,
915 cell-space-bottom-limit+ .code:n =
916     \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
917 cell-space-bottom-limit+ .value_required:n = true ,
918 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
919 cell-space-limits .meta:n =
920     {
921         cell-space-top-limit = #1 ,
922         cell-space-bottom-limit = #1 ,
923     } ,
924 cell-space-limits .value_required:n = true ,
925 cell-space-limits+ .meta:n =
926     {
927         cell-space-top-limit += #1 ,
928         cell-space-bottom-limit += #1 ,
929     } ,
930 cell-space-limits+ .value_required:n = true ,
931 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
932 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
933 light-syntax .code:n =
934     \bool_set_true:N \l_@@_light_syntax_bool
935     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
936 light-syntax .value_forbidden:n = true ,
937 light-syntax-expanded .code:n =
938     \bool_set_true:N \l_@@_light_syntax_bool
939     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
940 light-syntax-expanded .value_forbidden:n = true ,
941 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
942 end-of-row .value_required:n = true ,
943
944 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
945 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
946 last-row .int_set:N = \l_@@_last_row_int ,
947 last-row .default:n = -1 ,
948
949 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
950 code-for-first-col .value_required:n = true ,
951 code-for-first-col+ .code:n =
952     { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
953 code-for-first-col+ .value_required:n = true ,
954 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
955
956 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
957 code-for-last-col .value_required:n = true ,
958 code-for-last-col+ .code:n =
959     { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
960 code-for-last-col+ .value_required:n = true ,

```

```

961 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
962
963 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
964 code-for-first-row .value_required:n = true ,
965 code-for-first-row+ .code:n =
966   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
967 code-for-first-row+ .value_required:n = true ,
968 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
969
970 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
971 code-for-last-row .value_required:n = true ,
972 code-for-last-row+ .code:n =
973   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
974 code-for-last-row+ .value_required:n = true ,
975 code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
976
977 hlines .clist_set:N = \l_@@_hlines_clist ,
978 vlines .clist_set:N = \l_@@_vlines_clist ,
979 hlines .default:n = all ,
980 vlines .default:n = all ,
981 vlines-in-sub-matrix .code:n =
982   {
983     \tl_if_single_token:nTF { #1 }
984     {
985       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
986       { \@@_error:nn { Forbidden~letter } { #1 } }

```

We write directly a command for the automata which reads the preamble provided by the final user.

```

987     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
988   }
989   { \@@_error:n { One~letter~allowed } }
990 } ,
991 vlines-in-sub-matrix .value_required:n = true ,
992 hvlines .code:n =
993   {
994     \bool_set_true:N \l_@@_hvlines_bool
995     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
996     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
997   } ,
998 hvlines .value_forbidden:n = true ,
999 hvlines-except-borders .code:n =
1000   {
1001     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1002     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1003     \bool_set_true:N \l_@@_hvlines_bool
1004     \bool_set_true:N \l_@@_except_borders_bool
1005   } ,
1006 hvlines-except-borders .value_forbidden:n = true ,
1007 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

1008 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1009 renew-dots .value_forbidden:n = true ,
1010 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1011 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1012 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1013 create-extra-nodes .meta:n =
1014   { create-medium-nodes , create-large-nodes } ,
1015 left-margin .dim_set:N = \l_@@_left_margin_dim ,
1016 left-margin .default:n = \arraycolsep ,
1017 right-margin .dim_set:N = \l_@@_right_margin_dim ,
1018 right-margin .default:n = \arraycolsep ,

```



```

1019   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1020   margin .default:n = \arraycolsep ,
1021   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1022   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1023   extra-margin .meta:n =
1024     { extra-left-margin = #1 , extra-right-margin = #1 } ,
1025   extra-margin .value_required:n = true ,
1026   respect-arraystretch .code:n =
1027     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1028   respect-arraystretch .value_forbidden:n = true ,
1029   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1030   pgf-node-code .value_required:n = true
1031 }

```

We define a set of keys used by the environments of `nicematrix` (but not by the command `\NiceMatrixOptions`).

```

1032 \keys_define:nn { nicematrix / environments }
1033 {
1034   create-blocks-in-col .code:n = \@@_create_blocks_in_col:n { #1 } ,
1035   create-blocks-in-col .value_required:n = true ,
1036   corners .clist_set:N = \l_@@_corners_clist ,
1037   corners .default:n = { NW , SW , NE , SE } ,
1038   code-before .code:n =
1039     {
1040       \tl_if_empty:nF { #1 }
1041       {
1042         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1043         \bool_set_true:N \l_@@_code_before_bool
1044       }
1045     } ,
1046   code-before .value_required:n = true ,

```

The options `c`, `t` and `b` of the environment `{NiceArray}` have the same meaning as the option of the classical environment `{array}`.

```

1047   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1048   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1049   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
1050   baseline .tl_set:N = \l_@@_baseline_tl ,
1051   baseline .value_required:n = true ,
1052   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1053   \str_if_eq:eeTF { #1 } { auto }
1054   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1055   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1056   columns-width .value_required:n = true ,
1057   name .code:n =

```

We test whether we are in the measuring phase of an environment of `amsmath` (always loaded by `nicematrix`) because we want to avoid a fallacious message of duplicate name in this case.

```

1058   \legacy_if:nF { measuring@ }
1059   {
1060     \str_set:Ne \l_@@_name_str { #1 }
1061     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1062     { \@@_err_duplicate_names:n { #1 } }
1063     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1064   } ,
1065   name .value_required:n = true ,
1066   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1067   code-after .value_required:n = true ,
1068 }

```

```

1069 \cs_set:Npn \@@_err_duplicate_names:n #1
1070 { \@@_error:nn { Duplicate-name } { #1 } }
1071 \keys_define:nn { nicematrix / notes }
1072 {
1073   para .bool_set:N = \l_@@_notes_para_bool ,
1074   para .default:n = true ,
1075   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1076   code-before .value_required:n = true ,
1077   code-before+ .code:n =
1078     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1079   code-before+ .value_required:n = true ,
1080   code-before~+ .code:n =
1081     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1082   code-before~+ .value_required:n = true ,
1083   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1084   code-after .value_required:n = true ,
1085   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1086   bottomrule .default:n = true ,
1087   style .cs_set:Np = \@@_notes_style:n #1 ,
1088   style .value_required:n = true ,
1089   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1090   label-in-tabular .value_required:n = true ,
1091   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1092   label-in-list .value_required:n = true ,
1093   enumitem-keys .code:n =
1094     {
1095       \hook_gput_code:nnn { begindocument } { . }
1096       {
1097         \IfPackageLoadedT { enumitem }
1098           { \setlist* [ tabularnotes ] { #1 } }
1099       }
1100     } ,
1101   enumitem-keys .value_required:n = true ,
1102   enumitem-keys-para .code:n =
1103     {
1104       \hook_gput_code:nnn { begindocument } { . }
1105       {
1106         \IfPackageLoadedT { enumitem }
1107           { \setlist* [ tabularnotes* ] { #1 } }
1108       }
1109     } ,
1110   enumitem-keys-para .value_required:n = true ,
1111   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1112   detect-duplicates .default:n = true ,
1113   unknown .code:n =
1114     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1115 }
1116 \keys_define:nn { nicematrix / delimiters }
1117 {
1118   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1119   max-width .default:n = true ,
1120   color .tl_set:N = \l_@@_delimiters_color_tl ,
1121   color .value_required:n = true ,
1122 }

```

We begin the construction of the major sets of keys (used by the different user commands and environments).

```

1123 \keys_define:nn { nicematrix }
1124 {
1125   NiceMatrixOptions .inherit:n =
1126     { nicematrix / Global } ,
1127   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

1128 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1129 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1130 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1131 SubMatrix / rules .inherit:n = nicematrix / rules ,
1132 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1133 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1134 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1135 NiceMatrix .inherit:n =
1136 {
1137     nicematrix / Global ,
1138     nicematrix / environments ,
1139 } ,
1140 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1141 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1142 NiceTabular .inherit:n =
1143 {
1144     nicematrix / Global ,
1145     nicematrix / environments
1146 } ,
1147 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1148 NiceTabular / rules .inherit:n = nicematrix / rules ,
1149 NiceTabular / notes .inherit:n = nicematrix / notes ,
1150 NiceArray .inherit:n =
1151 {
1152     nicematrix / Global ,
1153     nicematrix / environments ,
1154 } ,
1155 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1156 NiceArray / rules .inherit:n = nicematrix / rules ,
1157 pNiceArray .inherit:n =
1158 {
1159     nicematrix / Global ,
1160     nicematrix / environments ,
1161 } ,
1162 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1163 pNiceArray / rules .inherit:n = nicematrix / rules ,
1164 }

```

We finalise the definition of the set of keys “`nicematrix / NiceMatrixOptions`” with the options specific to `\NiceMatrixOptions`.

```

1165 \keys_define:nn { nicematrix / NiceMatrixOptions }
1166 {
1167     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1168     delimiters / color .value_required:n = true ,
1169     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1170     delimiters / max-width .default:n = true ,
1171     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1172     delimiters .value_required:n = true ,
1173     width .dim_set:N = \l_@@_width_dim ,
1174     width .value_required:n = true ,
1175     last-col .code:n =
1176     \tl_if_empty:nF { #1 }
1177     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1178     \int_zero:N \l_@@_last_col_int ,
1179     small .bool_set:N = \l_@@_small_bool ,
1180     small .value_forbidden:n = true ,

```

With the option `renew-matrix`, the environment `{matrix}` of `amsmath` and its variants are redefined to behave like the environment `{NiceMatrix}` and its variants.

```

1181 renew-matrix .code:n = \@@_renew_matrix: ,
1182 renew-matrix .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```
1183     exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
```

If the option `columns-width` is used, all the columns will have the same width. In `\NiceMatrixOptions`, the special value `auto` is not available.

```
1184     columns-width .code:n =
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
1185     \str_if_eq:eeTF { #1 } { auto }
1186     { \@@_error:n { Option~auto~for~columns~width } }
1187     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

Usually, an error is raised when the user tries to give the same name to two distinct environments of `nicematrix` (these names are global and not local to the current TeX scope). However, the option `allow-duplicate-names` disables this feature.

```
1188     allow-duplicate-names .code:n =
1189     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1190     allow-duplicate-names .value_forbidden:n = true ,
1191     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1192     notes .value_required:n = true ,
1193     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1194     sub-matrix .value_required:n = true ,
1195     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1196     matrix / columns-type .value_required:n = true ,
1197     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1198     caption-above .default:n = true ,
1199     unknown .code:n =
1200     \@@_unknown_key:nn
1201     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1202     { Unknown-key-for-NiceMatrixOptions }
1203 }
```

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```
1204 \NewDocumentCommand \NiceMatrixOptions { m }
1205 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }
```

We finalise the definition of the set of keys “`nicematrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

```
1206 \keys_define:nn { nicematrix / NiceMatrix }
1207 {
1208     last-col .code:n = \tl_if_empty:nTF { #1 }
1209     {
1210         \bool_set_true:N \l_@@_last_col_without_value_bool
1211         \int_set:Nn \l_@@_last_col_int { -1 }
1212     }
1213     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1214     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1215     columns-type .value_required:n = true ,
1216     l .meta:n = { columns-type = l } ,
1217     r .meta:n = { columns-type = r } ,
1218     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1219     delimiters / color .value_required:n = true ,
1220     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1221     delimiters / max-width .default:n = true ,
1222     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1223     delimiters .value_required:n = true ,
1224     small .bool_set:N = \l_@@_small_bool ,
```

```

1225     small .value_forbidden:n = true ,
1226     unknown .code:n =
1227       \@@_unknown_key:nn
1228       { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1229       { Unknown-key-for-NiceMatrix }
1230   }

```

We finalise the definition of the set of keys “nicematrix / NiceArray” with the options specific to {NiceArray}.

```

1231 \keys_define:nn { nicematrix / NiceArray }
1232 {

```

In the environments {NiceArray} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.

```

1233     small .bool_set:N = \l_@@_small_bool ,
1234     small .value_forbidden:n = true ,
1235     last-col .code:n = \tl_if_empty:nF { #1 }
1236       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1237       \int_zero:N \l_@@_last_col_int ,
1238     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1239     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1240     unknown .code:n =
1241       \@@_unknown_key:nn
1242       { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1243       { Unknown-key-for-NiceArray }
1244   }

1245 \keys_define:nn { nicematrix / pNiceArray }
1246 {
1247     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1248     last-col .code:n = \tl_if_empty:nF { #1 }
1249       { \@@_error:n { last-col-non-empty-for-NiceArray } }
1250       \int_zero:N \l_@@_last_col_int ,
1251     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1252     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1253     delimiters / color .value_required:n = true ,
1254     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1255     delimiters / max-width .default:n = true ,
1256     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1257     delimiters .value_required:n = true ,
1258     small .bool_set:N = \l_@@_small_bool ,
1259     small .value_forbidden:n = true ,
1260     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1261     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1262     unknown .code:n =
1263       \@@_unknown_key:nn
1264       { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1265       { Unknown-key-for-NiceMatrix }
1266   }

```

We finalise the definition of the set of keys “nicematrix / NiceTabular” with the options specific to {NiceTabular}.

```

1267 \keys_define:nn { nicematrix / NiceTabular }
1268 {

```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```

1269     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1270       \bool_set_true:N \l_@@_width_used_bool ,
1271     width .value_required:n = true ,
1272     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1273     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,

```

```

1274     tabularnote .value_required:n = true ,
1275     caption .tl_set:N = \l_@@_caption_tl ,
1276     caption .value_required:n = true ,
1277     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1278     short-caption .value_required:n = true ,
1279     label .tl_set:N = \l_@@_label_tl ,
1280     label .value_required:n = true ,
1281     last-col .code:n = \tl_if_empty:nF { #1 }
1282         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1283         \int_zero:N \l_@@_last_col_int ,
1284     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1285     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1286     unknown .code:n =
1287         \@@_unknown_key:nn
1288         { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1289         { Unknown-key-for-NiceTabular }
1290 }

```

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs *key=value* between square brackets. Here is the corresponding set of keys. We *must* put the following instructions *after* the :

```
CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix
```

```

1291 \keys_define:nn { nicematrix / CodeAfter }
1292 {
1293     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1294     delimiters / color .value_required:n = true ,
1295     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1296     rules .value_required:n = true ,
1297     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1298     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1299     sub-matrix .value_required:n = true ,
1300     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1301 }

```

## 8 Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:-\@@_cell_end:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment `{array}`).

```

1302 \cs_new_protected:Npn \@@_cell_begin:
1303 {

```

`\g_@@_cell_after_hook_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box.

```
1304     \tl_gclear:N \g_@@_cell_after_hook_tl
```

At the beginning of the cell, we link `\CodeAfter` to a command which do begin with `\\` (whereas the standard version of `\CodeAfter` does not).

```
1305     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

The following link only to have a better error message when `\Hline` is used in another place than the beginning of a line.

```
1306     \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

We increment the LaTeX counter `jCol`, which is the counter of the columns.

```
1307     \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don't do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don't want to take into account.

Here is a version with the standard syntax of L3.

```
\int_compare:nNtT { \c@jCol } = { 1 }
  { \int_compare:nNtT \l_@@_first_col_int = { 1 } { \@@_begin_of_row: } }
```

We will use a version a little more efficient.

```
1308   \if_int_compare:w \c@jCol = \c_one_int
1309     \if_int_compare:w \l_@@_first_col_int = \c_one_int
1310       \@@_begin_of_row:
1311     \fi:
1312   \fi:
```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` is in the `\@@_cell_end:`.

```
1313   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1314   \@@_tuning_not_tabular_begin:
1315   \@@_tuning_exterior_rows:
1316   \g_@@_row_style_tl
1317 }
1318 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_last_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
  { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
}
```

We will use a version a little more efficient.

```
1319 \cs_new_protected:Npn \@@_tuning_first_last_row:
1320 {
1321   \if_int_compare:w \c@iRow = \c_zero_int
1322     \if_int_compare:w \c@jCol > \c_zero_int
1323       \l_@@_code_for_first_row_tl
1324       \xglobal \colorlet { nicematrix-first-row } { . }
1325     \fi:
1326   \else:
1327     \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1328   \fi:
1329 }
```

The following command will be nullified unless there is a last row and we know its value (*ie*: `\l_@@_lat_row_int > 0`).

```
\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNtT { \c@iRow } = { \l_@@_last_row_int }
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}
```

We will use a version a little more efficient.

```

1330 \cs_new_protected:Npn \@@_tuning_last_row:
1331 {
1332   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1333     \l_@@_code_for_last_row_tl
1334     \xglobal \colorlet { nicematrix-last-row } { . }
1335   \fi:
1336 }

```

A different value will be provided to the following commands when the key `small` is in force.

```

1337 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1338 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1339 {
1340   \m@th
1341   $ % $

```

A special value is provided by the following control sequence when the key `small` is in force.

```

1342   \@@_tuning_key_small:
1343 }
1344 \cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $

```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```

1345 \cs_new_protected:Npn \@@_begin_of_row:
1346 {
1347   \int_gincr:N \c@iRow
1348   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1349   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1350   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1351   \pgfpicture
1352   \pgfrememberpicturepositiononpagetrue
1353   \pgfcoordinate
1354     { \@@_env: - row - \int_use:N \c@iRow - base }
1355     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1356   \str_if_empty:NF \l_@@_name_str
1357     {
1358       \pgfnodealias
1359         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1360         { \@@_env: - row - \int_use:N \c@iRow - base }
1361     }
1362   \endpgfpicture
1363 }

```

Remark: If the key `create-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give information about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command. Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_update_for_first_and_last_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \dim_compare:nNnT
      { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
      { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
    \dim_compare:nNnT
      { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
      { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
  }
}

```



```

}
{
  \int_compare:nNt { \c@iRow } = { 1 }
  {
    \dim_compare:nNt
      { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
      { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
  }
}
}

```

We will use a version a little more efficient.

```

1364 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1365 {
1366   \if_int_compare:w \c@iRow = \c_zero_int
1367     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1368       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1369     \fi:
1370     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1371       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1372     \fi:
1373   \else:
1374     \if_int_compare:w \c@iRow = \c_one_int
1375       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1376         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1377       \fi:
1378     \fi:
1379   \fi:
1380 }

1381 \cs_new_protected:Npn \@@_rotate_cell_box:
1382 {
1383   \box_rotate:Nn \l_@@_cell_box { 90 }
1384   \bool_if:NTF \g_@@_rotate_c_bool
1385   {
1386     \hbox_set:Nn \l_@@_cell_box
1387     {
1388       \m@th
1389       $ % $
1390       \vcenter { \box_use:N \l_@@_cell_box }
1391       $ % $
1392     }
1393   }
1394   {
1395     \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
1396     {
1397       \vbox_set_top:Nn \l_@@_cell_box
1398       {
1399         \vbox_to_zero:n { }
1400         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1401         \box_use:N \l_@@_cell_box
1402       }
1403     }
1404   }
1405   \bool_gset_false:N \g_@@_rotate_bool
1406   \bool_gset_false:N \g_@@_rotate_c_bool
1407 }

```

Here is a version of the command `\@@_adjust_size_box:` with the syntax of standard L3.

```

\cs_new_protected:Npn \@@_adjust_size_box:
{
  \dim_compare:nNt { \g_@@_blocks_wd_dim } > { \c_zero_dim }

```

```

    {
      \dim_compare:nNnT \g_@@_blocks_wd_dim > { \box_wd:N \l_@@_cell_box }
      { \box_set_wd:Nn \l_@@_cell_box \g_@@_blocks_wd_dim }
      \dim_gzero:N \g_@@_blocks_wd_dim
    }
  \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
  {
    \dim_compare:nNnT \g_@@_blocks_dp_dim > { \box_dp:N \l_@@_cell_box }
    { \box_set_dp:Nn \l_@@_cell_box \g_@@_blocks_dp_dim }
    \dim_gzero:N \g_@@_blocks_dp_dim
  }
  \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
  {
    \dim_compare:nNnT \g_@@_blocks_ht_dim > { \box_ht:N \l_@@_cell_box }
    { \box_set_ht:Nn \l_@@_cell_box \g_@@_blocks_ht_dim }
    \dim_gzero:N \g_@@_blocks_ht_dim
  }
}

```

Here is a version slightly more efficient.

```

1408 \cs_set_protected:Npn \@@_adjust_size_box:
1409 {
1410   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1411     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1412       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1413     \fi:
1414     \global \g_@@_blocks_wd_dim = \c_zero_dim
1415   \fi:
1416   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1417     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1418       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1419     \fi:
1420     \global \g_@@_blocks_dp_dim = \c_zero_dim
1421   \fi:
1422   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1423     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1424       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1425     \fi:
1426     \global \g_@@_blocks_ht_dim = \c_zero_dim
1427   \fi:
1428 }
1429 \cs_new_protected:Npn \@@_cell_end:
1430 {

```

The following command is nullified in the tabulars.

```

1431   \@@_tuning_not_tabular_end:
1432   \hbox_set_end:
1433   \@@_cell_end_i:
1434 }

```

```

\cs_new_protected:Npn \@@_cell_end_i:
{
  \g_@@_cell_after_hook_tl
  \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
  \@@_adjust_size_box:
  \box_set_ht:Nn \l_@@_cell_box
  { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
  \box_set_dp:Nn \l_@@_cell_box
  { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
  \@@_update_max_cell_width:
  \@@_update_for_first_and_last_row:
  \bool_if:NTF \g_@@_empty_cell_bool
  { \box_use_drop:N \l_@@_cell_box }
  {

```

```

\bool_if:NTF \g_@@_not_empty_cell_bool
{ \@@_print_node_cell: }
{
  \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
  { \@@_print_node_cell: }
  { \box_use_drop:N \l_@@_cell_box }
}
}
\int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
{ \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
\bool_gset_false:N \g_@@_empty_cell_bool
\bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1435 \cs_new_protected:Npn \@@_cell_end_i:
1436 {

```

The token list `\g_@@_cell_after_hook_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

1437   \g_@@_cell_after_hook_tl
1438   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1439   \@@_adjust_size_box:
1440   \box_set_ht:Nn \l_@@_cell_box
1441   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1442   \box_set_dp:Nn \l_@@_cell_box
1443   { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

1444   \@@_update_max_cell_width:

```

The following computations are for the “first row” and the “last row”.

```

1445   \@@_update_for_first_and_last_row:

```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s difficult to determine whether a cell is empty. Up to now we use the following technique:

- for the columns of type p, m, b, V (of varwidth) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`
- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, `\llap`, `\clap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `\CodeAfter`); however, if `nullify-dots` is not in force, a phantom of `\ldots`, `\cdots`, `\vdots` is inserted and its width is not equal to zero; that’s why these commands raise a boolean `\g_@@_empty_cell_bool` and we begin by testing this boolean.

```

1446   \bool_if:NTF \g_@@_empty_cell_bool
1447   { \box_use_drop:N \l_@@_cell_box }
1448   {
1449     \bool_if:NTF \g_@@_not_empty_cell_bool
1450     { \@@_print_node_cell: }
1451     {
1452       \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim

```

```

1453         \@@_print_node_cell:
1454     \else:
1455         \box_use_drop:N \l_@@_cell_box
1456     \fi:
1457 }
1458 }
1459 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1460     \global \g_@@_col_total_int = \c@jCol
1461 \fi:
1462 \global \let \g_@@_empty_cell_bool \c_false_bool
1463 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1464 }

```

The following command will be nullified in our redefinition of `\multicolumn`.

```

\cs_new_protected:Npn \@@_update_max_cell_width:
{
    \dim_gset:Nn \g_@@_max_cell_width_dim
    { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
}

```

We will use the following version, slightly more efficient:

```

1465 \cs_new_protected:Npn \@@_update_max_cell_width:
1466 {
1467     \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1468         \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1469     \fi:
1470 }

```

The following variant of `\@@_cell_end:` is only for the columns of type `w{s}{...}` or `W{s}{...}` (which use the horizontal alignment key `s` of `\makebox`).

```

1471 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1472 {
1473     \@@_math_toggle:
1474     \hbox_set_end:
1475     \bool_if:NF \g_@@_rotate_bool
1476     {
1477         \hbox_set:Nn \l_@@_cell_box
1478         {
1479             \makebox [ \l_@@_col_width_dim ] [ s ]
1480             { \hbox_unpack_drop:N \l_@@_cell_box }
1481         }
1482     }
1483     \@@_cell_end_i:
1484 }

```

```

1485 \pgfset
1486 {
1487     nicematrix / cell-node /.style =
1488     {
1489         inner-sep = \c_zero_dim ,
1490         minimum-width = \c_zero_dim
1491     }
1492 }

```

In the cells of a column of type `S` (of `siunitx`), we have to wrap the command `\@@_node_cell:` inside a command of `siunitx` to enforce the correct horizontal alignment. In the cells of the columns with other columns type, we don't have to do that job. That's why we create a socket with its default plug (`identity`) and a plug when we have to do the wrapping.

```

1493 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }

```

```

1494 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1495 {
1496   \use:c
1497   {
1498     __siunitx_table_align_
1499     \bool_if:NTF \l__siunitx_table_text_bool
1500     { \l__siunitx_table_align_text_tl }
1501     { \l__siunitx_table_align_number_tl }
1502     :n
1503   }
1504   { #1 }
1505 }

```

Now, a socket which deal with `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```

1506 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1507 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1508 {
1509   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1510   \hbox:n
1511   {
1512     \pgfsys@markposition
1513     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1514   }
1515   #1
1516   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1517   \hbox:n
1518   {
1519     \pgfsys@markposition
1520     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1521   }
1522 }

1523 \cs_new_protected:Npn \@@_print_node_cell:
1524 {
1525   \socket_use:nn { nicematrix / siunitx-wrap }
1526   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1527 }

```

The following command creates the PGF name of the node with, of course, `\l_@@_cell_box` as the content.

```

1528 \cs_new_protected:Npn \@@_node_cell:
1529 {
1530   \pgfpicture
1531   \pgfsetbaseline \c_zero_dim
1532   \pgfrememberpicturepositiononpagetrue
1533   \pgfset { nicematrix / cell-node }
1534   \pgfnode
1535   { rectangle }
1536   { base }
1537   {

```

The following instruction `\set@color` has been added on 2022/10/06. It’s necessary only with Xe-LaTeX and not with the other engines (we don’t know why).

```

1538   \sys_if_engine_xetex:T { \set@color }
1539   \box_use:N \l_@@_cell_box
1540 }
1541 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1542 { \l_@@_pgf_node_code_tl }
1543 \str_if_empty:NF \l_@@_name_str

```

```

1544 {
1545   \pgfnodealias
1546   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1547   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1548 }
1549 \endpgfpicture
1550 }

```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots[color=red] & & 
\end{pNiceMatrix}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \end{pmatrix}$$

the content of `\g_@@_Cdots_lines_tl` will be:

```

\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{color=red}

```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions (with consequences for the parallelisation of the diagonal lines).

```

1551 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1552 {
1553   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1554   { g_@@_ #2 _ lines _ tl }
1555   {
1556     \use:c { @@ _ draw _ #2 : nnn }
1557     { \int_use:N \c@iRow }
1558     { \int_use:N \c@jCol }
1559     { \exp_not:n { #3 } }
1560   }
1561 }

1562 \cs_new_protected:Npn \@@_array:n
1563 {
1564   \dim_set:Nn \col@sep
1565   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1566   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1567   { \def \@halignto { } }
1568   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

If `colortbl` is loaded, `\@tabarray` has been redefined to incorporate `\CT@start`.

```

1569   \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
1570   [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1571   }
1572 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, it seems that RevTeX goes on with a redefinition of `array` which uses `\ialign`.

```

1573 \bool_if:NTF \c_@@_revtex_bool
1574 { \cs_new_eq:NN \@@_old_ialign: \ialign }

```

We use here a `\cs_set_eq:cN` instead of a `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```
1575 { \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign }
```

The following command creates a row node (and not a row of nodes!).

```
1576 \cs_new_protected:Npn \@@_create_row_node:
1577 {
1578   \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1579   {
1580     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1581     \@@_create_row_node_i:
1582   }
1583 }
1584 \cs_new_protected:Npn \@@_create_row_node_i:
1585 {
```

The `\hbox:n` (or `\hbox`) is mandatory.

```
1586   \hbox
1587   {
1588     \bool_if:NT \l_@@_code_before_bool
1589     {
1590       \vtop
1591       {
1592         \skip_vertical:N 0.5\arrayrulewidth
1593         \pgfsys@markposition
1594         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1595         \skip_vertical:N -0.5\arrayrulewidth
1596       }
1597     }
1598     \pgfpicture
1599     \pgfrememberpicturerepositiononpagetrue
1600     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1601     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1602     \str_if_empty:NF \l_@@_name_str
1603     {
1604       \pgfnodealias
1605       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1606       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1607     }
1608     \endpgfpicture
1609   }
1610 }
```

```
1611 \cs_new_protected:Npn \@@_in_everycr:
1612 {
1613   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1614   \tbl_update_cell_data_for_next_row:
1615   \int_gzero:N \c@jCol
1616   \bool_gset_false:N \g_@@_after_col_zero_bool
1617   \bool_if:NF \g_@@_row_of_col_done_bool
1618   {
1619     \@@_create_row_node:
```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for these rules (the rules will be drawn by PGF).

```
1620   \clist_if_empty:NF \l_@@_hlines_clist
1621   {
1622     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1623     {
1624       \clist_if_in:NnT
1625       \l_@@_hlines_clist
1626       { \int_eval:n { \c@iRow + 1 } }
1627     }
1628   }
```

The counter `\c@iRow` has the value  $-1$  only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

```

1629         \int_compare:nNnT { \c@iRow } > { -1 }
1630         {
1631             \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1632             { \hrule height \arrayrulewidth width \c_zero_dim }
1633         }
1634     }
1635 }
1636 }
1637 }

```

When the key `renew-dots` is used, the following code will be executed.

```

1638 \cs_set_protected:Npn \@@_renew_dots:
1639 {
1640     \cs_set_eq:NN \ldots \@@_Ldots:
1641     \cs_set_eq:NN \cdots \@@_Cdots:
1642     \cs_set_eq:NN \vdots \@@_Vdots:
1643     \cs_set_eq:NN \ddots \@@_Ddots:
1644     \cs_set_eq:NN \iddots \@@_Iddots:
1645     \cs_set_eq:NN \dots \@@_Ldots:
1646     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1647 }

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition <sup>5</sup>.

```

1648 \hook_gput_code:nnn { begindocument } { . }
1649 {
1650     \IfPackageLoadedTF { booktabs }
1651     {
1652         \cs_new_protected:Npn \@@_patch_booktabs:
1653         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1654     }
1655     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1656 }

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`<sup>6</sup> and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That’s why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that’s why we do it in the `\ialign`.

```

1657 \cs_new_protected:Npn \@@_some_initialization:
1658 {
1659     \@@_everycr:
1660     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1661     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1662     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1663     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1664     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1665     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1666 }

```

<sup>5</sup>cf. `\nicematrix@redefine@check@rerun`

<sup>6</sup>The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.



`\@@_pre_array_after_CodeBefore:` will be executed in `\@@_pre_array:` *after* the execution of the `\CodeBefore`. It contains all the code before the beginning of the construction of `\l_@@_the_array_box`.

```
1667 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1668 {
```

The value of `\g_@@_pos_of_blocks_seq` has been written on the aux file and loaded before the (potential) execution of the `\CodeBefore`.

Now, we reinitialize that variable with the content of `\g_@@_future_pos_of_blocks_seq` because the main blocks will be added in `\g_@@_pos_of_blocks_seq` during the construction of the array.

```
1669 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1670 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1671 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq
```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```
1672 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value  $-2$  is important.

The total weight of the letters X in the preamble of the array.

```
1673 \fp_gzero:N \g_@@_total_X_weight_fp
1674 \bool_gset_false:N \g_@@_V_of_X_bool

1675 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1676 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1677 \@@_patch_booktabs:
1678 \box_clear_new:N \l_@@_cell_box
1679 \normalbaselines
```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```
1680 \bool_if:NT \l_@@_small_bool
1681 {
1682     \def \arraystretch { 0.47 }
1683     \dim_set:Nn \arraycolsep { 1.45 pt }
```

By default, `\@@_tuning_key_small:` is no-op.

```
1684 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1685 }
```

The boolean `\g_@@_create_cell_nodes_bool` corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
1686 \bool_if:NT \g_@@_create_cell_nodes_bool
1687 {
1688     \tl_put_right:Nn \@@_begin_of_row:
1689     {
1690         \pgfsys@markposition
1691         { \@@_env: - row - \int_use:N \c@iRow - base }
1692     }
1693     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1694 }
```

The environment `{array}` uses internally the command `\ar@ialign`. We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1695 \bool_if:NF \c_@@_revtex_bool
1696 {
1697   \def \ar@ialign
1698   {
1699     \tbl_init_cell_data_for_table:
1700     \@@_some_initialization:
1701     \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With that programming, we will have, in the cells of the array, a clean version of `\ar@ialign`. We use `\cs_set_eq:Nc` instead of `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```

1702     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1703     \halign
1704   }
1705 }

```

It seems that there is a problem when `nicematrix` is used with in `revtex4-2` with the package `colortbl` loaded. The following code prevent that problem but it does *not* treat the actual problem! It's only a patch *ad hoc*.

That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```

1706 \bool_if:NT \c_@@_revtex_bool
1707 {
1708   \IfPackageLoadedT { colortbl }
1709   { \cs_set_protected:Npn \CT@setup { } }
1710 }

```

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1711 \cs_set_eq:NN \@@_old_ldots: \ldots
1712 \cs_set_eq:NN \@@_old_cdots: \cdots
1713 \cs_set_eq:NN \@@_old_vdots: \vdots
1714 \cs_set_eq:NN \@@_old_ddots: \ddots
1715 \cs_set_eq:NN \@@_old_iddots: \iddots
1716 \bool_if:NTF \l_@@_standard_cline_bool
1717 { \cs_set_eq:NN \cline \@@_standard_cline: }
1718 { \cs_set_eq:NN \cline \@@_cline: }
1719 \cs_set_eq:NN \Ldots \@@_Ldots:
1720 \cs_set_eq:NN \Cdots \@@_Cdots:
1721 \cs_set_eq:NN \Vdots \@@_Vdots:
1722 \cs_set_eq:NN \Ddots \@@_Ddots:
1723 \cs_set_eq:NN \Iddots \@@_Iddots:
1724 \cs_set_eq:NN \Hline \@@_Hline:
1725 \cs_set_eq:NN \Hspace \@@_Hspace:
1726 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1727 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1728 \cs_set_eq:NN \Block \@@_Block:
1729 \cs_set_eq:NN \rotate \@@_rotate:
1730 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1731 \cs_set_eq:NN \dotfill \@@_dotfill:
1732 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1733 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1734 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1735 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1736 \cs_set_eq:NN \TopRule \@@_TopRule
1737 \cs_set_eq:NN \MidRule \@@_MidRule
1738 \cs_set_eq:NN \BottomRule \@@_BottomRule
1739 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1740 \cs_set_eq:NN \Hbrace \@@_Hbrace

```

```

1741 \cs_set_eq:NN \Vbrace \@@_Vbrace
1742 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1743 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1744 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1745 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1746 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1747 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1748 \int_if_zero:nTF \l_@@_first_row_int
1749 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1750 {
1751   \int_compare:nNnT { \l_@@_last_row_int } > { \c_zero_int }
1752   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1753 }
1754 \bool_if:NT \l_@@_renew_dots_bool { \@@_renew_dots: }

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1755 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1756 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1757 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1758 \@@_revert_colortbl:

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the aux file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1759 \tl_if_exist:NT \l_@@_note_in_caption_tl
1760 {
1761   \tl_if_empty:NF \l_@@_note_in_caption_tl
1762   {
1763     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1764     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1765   }
1766 }

```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with  $n > 1$  is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of  $n$ ) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```

1767 \seq_gclear:N \g_@@_multicolumn_cells_seq
1768 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1769 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1770 \int_gzero:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:` executed at the beginning of each cell.

```

1771 \int_gzero:N \g_@@_col_total_int
1772 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1773 \bool_gset_false:N \g_@@_last_col_found_bool

```

During the construction of the array, the instructions `\Cdots`, `\Ldots`, etc. will be written in token lists `\g_@@_Cdots_lines_tl`, etc. which will be executed after the construction of the array.

```

1774 \tl_gclear_new:N \g_@@_Cdots_lines_tl

```

```

1775 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1776 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1777 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1778 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1779 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1780 \tl_gclear:N \g_nicematrix_code_before_tl
1781 \tl_gclear:N \g_@@_pre_code_before_tl

```

We compute the width of both delimiters. We remind that, when the environment `{NiceArray}` is used, it's possible to specify the delimiters in the preamble (eg `[ccc]`).

```

1782 \dim_zero_new:N \l_@@_left_delim_dim
1783 \dim_zero_new:N \l_@@_right_delim_dim
1784 \bool_if:NTF \g_@@_delims_bool
1785 {

```

The command `\bBigg@` is a command of `amsmath`.

```

1786 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1787 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1788 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1789 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1790 }
1791 {
1792 \dim_gset:Nn \l_@@_left_delim_dim
1793 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1794 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1795 }
1796 }

```

This is the end of `\@@_pre_array_after_CodeBefore:`.

The command `\@@_pre_array:` will be executed after analysis of the keys of the environment. If will, in particular, read the potential informations written on the `aux` file.

```

1797 \cs_new_protected:Npn \@@_pre_array:
1798 {
1799 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1800 \int_gzero_new:N \c@iRow
1801 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1802 \int_gzero_new:N \c@jCol

```

We give values to the LaTeX counters `iRow` and `jCol`. We remind that before and after the main array (in particular in the `\CodeBefore` and the `\CodeAfter`, they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1803 \int_compare:nNt \l_@@_last_row_int > 0
1804 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1805 \int_compare:nNt \l_@@_last_col_int > 0
1806 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1807 \bool_if:NT \g_@@_aux_found_bool
1808 {
1809 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1810 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1811 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1812 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1813 }

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_col_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-col` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

1814 \int_compare:nNt { \l_@@_last_row_int } = { -1 }
1815 {
1816   \bool_set_true:N \l_@@_last_row_without_value_bool
1817   \bool_if:NT \g_@@_aux_found_bool
1818   { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1819 }
1820 \int_compare:nNt { \l_@@_last_col_int } = { -1 }
1821 {
1822   \bool_if:NT \g_@@_aux_found_bool
1823   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1824 }

```

If there is an exterior row, we patch a command used in `\@@_cell_begin:` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1825 \int_compare:nNt { \l_@@_last_row_int } > { -2 }
1826 {
1827   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1828   {
1829     \dim_compare:nNt { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1830     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1831     \dim_compare:nNt { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1832     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1833   }
1834 }

1835 \seq_gclear:N \g_@@_cols_vlism_seq
1836 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore.`

```

1837 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

The code in `\@@_pre_array_after_CodeBefore:` is used only here.

```

1838 \@@_pre_array_after_CodeBefore:

```

Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `$` also).

```

1839 \hbox_set:Nw \l_@@_the_array_box
1840 \skip_horizontal:N \l_@@_left_margin_dim
1841 \skip_horizontal:N \l_@@_extra_left_margin_dim
1842 \UseTaggingSocket { tbl / hmode / begin }

```

The following code is a workaround to specify to the tagging system that the following code is *fake math* (it raises `\l__math_fakemath_bool` in recent versions of LaTeX).

```

1843 \m@th
1844 $ % $
1845 \bool_if:NTF \l_@@_light_syntax_bool
1846 { \use:c { @@-light-syntax } }
1847 { \use:c { @@-normal-syntax } }
1848 }

```

The following command `\@@_CodeBefore_Body:w` will be used when the keyword `\CodeBefore` is present at the beginning of the environment.

```

1849 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1850 {
1851   \tl_set:Nn \l_tmpa_tl { #1 }
1852   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1853   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1854   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1855   \bool_set_true:N \l_@@_code_before_bool

```

We go on with `\@@_pre_array`: which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```
1856 \@@_pre_array:
1857 }
```

## 9 The `\CodeBefore`

```
1858 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
```

The following command will be executed if the `\CodeBefore` has to be actually executed (that command will be used only once and is present alone only for legibility).

```
1859 \cs_new_protected:Npn \@@_pre_code_before:
1860 {
```

We will create all the `col` nodes and `row` nodes with the information written in the aux file. You use the technique described in the page 1247 of `pgfmanual.pdf`, version 3.1.10.

```
1861 \pgfsys@markposition { \@@_env: - position }
1862 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1863 \pgfpicture
1864 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1865 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1866 {
1867   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1868   \pgfcoordinate { \@@_env: - row - ##1 }
1869   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1870 }
```

Now, the recreation of the `col` nodes.

```
1871 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1872 {
1873   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1874   \pgfcoordinate { \@@_env: - col - ##1 }
1875   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1876 }
```

Now, the creation of the `cell` nodes (`i-j`), and, maybe also the “medium nodes” and the “large nodes”.

```
1877 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1878 \endpgfpicture
```

Now, you recreate the diagonal nodes by using the `row` nodes and the `col` nodes.

```
1879 \@@_create_diag_nodes:
```

Now, the recreation of the nodes of the blocks *which have a name*.

```
1880 \@@_create_blocks_nodes:
1881 \IfPackageLoadedT { tikz }
1882 {
1883   \tikzset
1884   {
1885     every~picture / .style =
1886     { overlay , name~prefix = \@@_env: - }
1887   }
1888 }
1889 \cs_set_eq:NN \cellcolor \@@_cellcolor
1890 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1891 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1892 \cs_set_eq:NN \rowcolor \@@_rowcolor
1893 \cs_set_eq:NN \rowcolors \@@_rowcolors
1894 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
```

```

1895 \cs_set_eq:NN \arraycolor \@@_arraycolor
1896 \cs_set_eq:NN \columncolor \@@_columncolor
1897 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1898 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1899 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1900 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1901 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1902 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1903 }

```

```

1904 \cs_new_protected:Npn \@@_exec_code_before:
1905 {

```

We mark the cells which are in the (empty) corners because those cells must not be colored. We should try to find a way to detected whether we actually have coloring instructions to execute...

```

1906 \clist_map_inline:Nn \l_@@_corners_cells_clist
1907 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1908 \seq_gclear_new:N \g_@@_colors_seq

```

The sequence `\g_@@_colors_seq` will always contain as first element the special color `nocolor`: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of `nicematrix`.

```

1909 \@@_add_to_colors_seq:nn { { nocolor } } { }
1910 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1911 \group_begin:

```

We compose the `\CodeBefore` in math mode in order to nullify the spaces put by the user between instructions in the `\CodeBefore`.

```

1912 \if_mode_math:
1913 \@@_exec_code_before_i:
1914 \else:
1915 $ % $
1916 \@@_exec_code_before_i:
1917 $ % $
1918 \fi:
1919 \group_end:
1920 }

```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `<` (de code ASCII 60) and `>` are activated and TikZ is not able to solve the problem (even with the TikZ library `babel`).

```

1921 \cs_new_protected:Npn \@@_exec_code_before_i:
1922 {
1923 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1924 { \@@_rescan_for_spanish:N \l_@@_code_before_tl }

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```

1925 \exp_last_unbraced:No \@@_CodeBefore_keys:
1926 \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1927 \@@_actually_color:
1928 \l_@@_code_before_tl
1929 \q_stop
1930 }

```

```

1931 \keys_define:nn { nicematrix / CodeBefore }
1932 {

```

```

1933     create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1934     create-cell-nodes .default:n = true ,
1935     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1936     sub-matrix .value_required:n = true ,
1937     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1938     delimiters / color .value_required:n = true ,
1939     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1940 }

1941 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1942 {
1943     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1944     \@@_CodeBefore:w
1945 }

```

We have extracted the options of the keyword `\CodeBefore` in order to see whether the key `create-cell-nodes` has been used. Now, you can execute the rest of the `\CodeBefore`, excepted, of course, if we are in the first compilation.

```

1946 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1947 {
1948     \bool_if:NTF \g_@@_aux_found_bool
1949     {
1950         \@@_pre_code_before:
1951         \legacy_if:nF { measuring@ } { #1 }
1952     }

```

If we are in the first compilation, you won't really execute the `\CodeBefore` but we have to execute some instructions of creation of PGF/TikZ pictures in order to have the correct `aux` file in the next run (hence, we avoid to “lose” a run).

```

1953     {
1954         \pgfsys@markposition { \@@_env: - position }
1955         \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1956         \pgfpicture
1957         \pgf@relevantforpicturesizefalse
1958         \endpgfpicture

```

The following picture corresponds to `\@@_create_diag_nodes:`

```

1959         \pgfpicture
1960         \pgfrememberpicturepositiononpagetrue
1961         \endpgfpicture

```

The following picture corresponds to `\@@_create_blocks_nodes:`

```

1962         \pgfpicture
1963         \pgf@relevantforpicturesizefalse
1964         \pgfrememberpicturepositiononpagetrue
1965         \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1966         \pgfpicture
1967         \pgf@relevantforpicturesizefalse
1968         \endpgfpicture
1969     }
1970 }

```

By default, if the user uses the `\CodeBefore`, only the `col` nodes, `row` nodes and `diag` nodes are available in that `\CodeBefore`. With the key `create-cell-nodes`, the cell nodes, that is to say the nodes of the form `(i-j)` (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

```

1971 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1972 {
1973     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1974     {
1975         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1976         \pgfcoordinate { \@@_env: - row - ##1 - base }
1977         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1978         \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }

```



```

1979     {
1980         \cs_if_exist:cT
1981         { pgf @ sys @ pdf @ mark @ pos @ \@_env: - ##1 - #####1 - NW }
1982         {
1983             \pgfsys@getposition
1984             { \@_env: - ##1 - #####1 - NW }
1985             \@_node_position:
1986             \pgfsys@getposition
1987             { \@_env: - ##1 - #####1 - SE }
1988             \@_node_position_i:
1989             \@_pgf_rect_node:nnn
1990             { \@_env: - ##1 - #####1 }
1991             { \pgfpointdiff \@_picture_position: \@_node_position: }
1992             { \pgfpointdiff \@_picture_position: \@_node_position_i: }
1993         }
1994     }
1995 }
1996 \@@_create_extra_nodes:
1997 \@@_create_aliases_last:
1998 }

1999 \cs_new_protected:Npn \@@_create_aliases_last:
2000 {
2001     \int_step_inline:nn { \c@iRow }
2002     {
2003         \pgfnodealias
2004         { \@_env: - ##1 - last }
2005         { \@_env: - ##1 - \int_use:N \c@jCol }
2006     }
2007     \int_step_inline:nn { \c@jCol }
2008     {
2009         \pgfnodealias
2010         { \@_env: - last - ##1 }
2011         { \@_env: - \int_use:N \c@iRow - ##1 }
2012     }
2013     \pgfnodealias
2014     { \@_env: - last - last }
2015     { \@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2016 }

2017 \cs_new_protected:Npn \@@_create_blocks_nodes:
2018 {
2019     \pgfpicture
2020     \pgf@relevantforpicturesizefalse
2021     \pgfrememberpicturepositiononpagetrue
2022     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2023     { \@@_create_one_block_node:nnnnn ##1 }
2024     \endpgfpicture
2025 }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.<sup>7</sup>

```

2026 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2027 {
2028     \tl_if_empty:nF { #5 }
2029     {
2030         \@@_qpoint:n { col - #2 }
2031         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2032         \@@_qpoint:n { #1 }

```

<sup>7</sup>Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

2033 \dim_set_eq:NN \l_tmpb_dim \pgf@y
2034 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2035 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2036 \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2037 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
2038 \@@_pgf_rect_node:nnnnn
2039 { \@@_env: - #5 }
2040 { \dim_use:N \l_tmpa_dim }
2041 { \dim_use:N \l_tmpb_dim }
2042 { \dim_use:N \l_@@_tmpc_dim }
2043 { \dim_use:N \l_@@_tmpd_dim }
2044 }
2045 }

2046 \cs_new_protected:Npn \@@_patch_for_revtext:
2047 {
2048 \cs_set_eq:NN \@addamp \@addamp@LaTeX
2049 \cs_set_eq:NN \@array \@array@array
2050 \cs_set_eq:NN \@tabular \@tabular@array
2051 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
2052 \cs_set_eq:NN \array \array@array
2053 \cs_set_eq:NN \endarray \endarray@array
2054 \cs_set:Npn \endtabular { \endarray $\egroup } % $
2055 \cs_set_eq:NN \@mkpream \@mkpream@array
2056 \cs_set_eq:NN \@classx \@classx@array
2057 \cs_set_eq:NN \insert@column \insert@column@array
2058 \cs_set_eq:NN \@arraycr \@arraycr@array
2059 \cs_set_eq:NN \@xarraycr \@xarraycr@array
2060 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
2061 }

```

## 10 The environment {NiceArrayWithDelims}

```

2062 \NewDocumentEnvironment { NiceArrayWithDelims }
2063 { m m O { } m ! O { } t \CodeBefore }
2064 {
2065 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
2066 \@@_provide_pgfsyspdfmark:
2067 \bool_if:NT \g_@@_footnote_bool { \savenotes }

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

2068 \bgroup

2069 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2070 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2071 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2072 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2073 \int_gzero:N \g_@@_block_box_int
2074 \dim_gzero:N \g_@@_width_last_col_dim
2075 \dim_gzero:N \g_@@_width_first_col_dim
2076 \bool_gset_false:N \g_@@_row_of_col_done_bool
2077 \str_if_empty:NT \g_@@_name_env_str
2078 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2079 \bool_if:NTF \l_@@_tabular_bool
2080 { \mode_leave_vertical: }
2081 { \@@_test_if_math_mode: }
2082 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2083 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>8</sup>. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```
2084 \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@
```

We deactivate TikZ externalization because we will use PGF pictures with the options `overlay` and `remember picture` (or equivalent forms). We deactivate with `\tikzexternaldisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```
2085 \cs_if_exist:NT \tikz@library@external@loaded
2086 {
2087     \tikzexternaldisable
2088     \cs_if_exist:NT \ifstandalone
2089     { \tikzset { external / optimize = false } }
2090 }
```

We increment the counter `\g_@@_env_int` which counts the environments of the package.

```
2091 \int_gincr:N \g_@@_env_int
2092 \bool_if:NF \l_@@_block_auto_columns_width_bool
2093 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

The sequence `\g_@@_blocks_seq` will contain the characteristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks.

```
2094 \seq_gclear:N \g_@@_blocks_seq
2095 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

In fact, the sequence `\g_@@_pos_of_blocks_seq` will also contain the positions of the cells with a `\diagbox` and the `\multicolumn`.

```
2096 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2097 \seq_gclear:N \g_@@_pos_of_xdots_seq
2098 \tl_gclear_new:N \g_@@_code_before_tl
2099 \tl_gclear:N \g_@@_row_style_tl
```

We load all the information written in the aux file during previous compilations corresponding to the current environment.

```
2100 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2101 {
2102     \bool_gset_true:N \g_@@_aux_found_bool
2103     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2104 }
2105 { \bool_gset_false:N \g_@@_aux_found_bool }
```

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```
2106 \tl_gclear:N \g_@@_aux_tl
2107 \tl_if_empty:NF \g_@@_code_before_tl
2108 {
2109     \bool_set_true:N \l_@@_code_before_bool
2110     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2111 }
2112 \tl_if_empty:NF \g_@@_pre_code_before_tl
2113 { \bool_set_true:N \l_@@_code_before_bool }
```

The set of keys is not exactly the same for `{NiceArray}` and for the variants of `{NiceArray}` (`{pNiceArray}`, `{bNiceArray}`, etc.) because, for `{NiceArray}`, we have the options `t`, `c`, `b` and `baseline`.

```
2114 \bool_if:NTF \g_@@_delims_bool
2115 { \keys_set:nn { nicematrix / pNiceArray } }
2116 { \keys_set:nn { nicematrix / NiceArray } }
2117 { #3 , #5 }
```

---

<sup>8</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```
2118 \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302
```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type “t \CodeBefore”, we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It’s the job that will do the command \@@\_CodeBefore\_Body:w. After that job, the command \@@\_CodeBefore\_Body:w will go on with \@@\_pre\_array:.

```
2119 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2120 }
```

Now, the second part of the environment {NiceArrayWithDelims}.

```
2121 {
2122   \bool_if:nTF \l_@@_light_syntax_bool
2123     { \use:c { end @@-light-syntax } }
2124     { \use:c { end @@-normal-syntax } }
2125   $ % $
2126   \skip_horizontal:N \l_@@_right_margin_dim
2127   \skip_horizontal:N \l_@@_extra_right_margin_dim
2128   \hbox_set_end:
2129   \UseTaggingSocket { tbl / hmode / end }
```

End of the construction of the array (in the box \l\_@@\_the\_array\_box).

If the user has used the key width without any column X, we raise an error.

```
2130 \bool_if:NT \l_@@_width_used_bool
2131 {
2132   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2133     { \@@_error_or_warning:n { width-without-X-columns } }
2134 }
```

Now, if there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, l\_@@\_X\_columns\_dim will be the width of a column of weight 1.0. For a X-column of weight  $x$ , the width will be  $\text{l\_@@\_X\_columns\_dim}$  multiplied by  $x$ .

```
2135 \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2136   { \@@_compute_width_X: }
```

If the user has used the key last-row with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```
2137 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2138 {
2139   \bool_if:NF \l_@@_last_row_without_value_bool
2140   {
2141     \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2142     {
2143       \@@_error:n { Wrong-last-row }
2144       \int_set_eq:NN \l_@@_last_row_int \c@iRow
2145     }
2146   }
2147 }
```

Now, the definition of \c@jCol and \g\_@@\_col\_total\_int changes: \c@jCol will be the number of columns without the “last column”; \g\_@@\_col\_total\_int will be the number of columns with this “last column”.<sup>9</sup>

```
2148 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2149 \bool_if:NTF \g_@@_last_col_found_bool
2150   { \int_gdecr:N \c@jCol }
2151   {
2152     \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2153     { \@@_error:n { last-col-not-used } }
```

---

<sup>9</sup>We remind that the potential “first column” (exterior) has the number 0.

2154 }

We fix also the value of `\c@iRow` and `\g_@@_row_total_int` with the same principle.

```
2155 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2156 \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2157 { \int_gdecr:N \c@iRow }
```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 95).

```
2158 \int_if_zero:nT { \l_@@_first_col_int }
2159 { \skip_horizontal:N \g_@@_width_first_col_dim }
```

The construction of the real box is different whether we have delimiters to put.

```
2160 \bool_if:nTF { ! \g_@@_delims_bool }
2161 {
2162   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2163   { \@@_use_arraybox_with_notes_c: }
2164   {
2165     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2166     { \@@_use_arraybox_with_notes_b: }
2167     { \@@_use_arraybox_with_notes: }
2168   }
2169 }
```

Now, in the case of an environment with delimiters. We compute `\l_tmpa_dim` which is the total height of the “first row” above the array (when the key `first-row` is used).

```
2170 {
2171   \int_if_zero:nTF { \l_@@_first_row_int }
2172   {
2173     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2174     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2175   }
2176   { \dim_zero:N \l_tmpa_dim }
```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.<sup>10</sup>

```
2177 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2178 {
2179   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2180   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2181 }
2182 { \dim_zero:N \l_tmpb_dim }
2183 \hbox_set:Nn \l_tmpa_box
2184 {
2185   \m@th
2186   $ % $
2187   \@@_color:o \l_@@_delimiters_color_tl
2188   \exp_after:wN \left \g_@@_left_delim_tl
2189   \vcenter
2190   {
```

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```
2191   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2192   \hbox
2193   {
2194     \bool_if:NTF \l_@@_tabular_bool
2195     { \skip_horizontal:n { - \tabcolsep } }
2196     { \skip_horizontal:n { - \arraycolsep } }
2197     \@@_use_arraybox_with_notes_c:
2198     \bool_if:NTF \l_@@_tabular_bool
```

<sup>10</sup>A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

2199         { \skip_horizontal:n { - \tabcolsep } }
2200         { \skip_horizontal:n { - \arraycolsep } }
2201     }

```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`).

```

2202         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2203     }
2204     \exp_after:wN \right \g_@@_right_delim_tl
2205     $ % $
2206 }

```

Now, the box `\l_tmpa_box` is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

2207     \bool_if:NTF \l_@@_delimiters_max_width_bool
2208     {
2209         \@@_put_box_in_flow_bis:nn
2210         { \g_@@_left_delim_tl }
2211         { \g_@@_right_delim_tl }
2212     }
2213     \@@_put_box_in_flow:
2214 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 96).

```

2215     \bool_if:NT \g_@@_last_col_found_bool
2216     { \skip_horizontal:N \g_@@_width_last_col_dim }
2217     \bool_if:NT \l_@@_preamble_bool
2218     {
2219         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2220         { \@@_err_columns_not_used: }
2221     }
2222     \@@_after_array:

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

2223     \egroup

```

We write on the aux file all the information corresponding to the current environment.

```

2224     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2225     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2226     \iow_now:Ne \@mainaux
2227     {
2228         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2229         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2230         { \exp_not:o \g_@@_aux_tl }
2231     }
2232     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2233     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2234 }

```

This is the end of the environment `{NiceArrayWithDelims}`.

```

2235     \cs_new_protected:Npn \@@_err_columns_not_used:
2236     {
2237         \@@_warning:n { columns~not~used }
2238         \cs_gset:Npn \@@_err_columns_not_used: { }
2239     }

```

The following command will be used only once. We have written that command for legibility. If there is at least one  $X$ -column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `\l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a  $X$ -column of weight  $x$ , the width will be `\l_@@_X_columns_dim` multiplied by  $x$ .

```

2240 \cs_new_protected:Npn \@@_compute_width_X:
2241 {
2242   \tl_gput_right:Ne \g_@@_aux_tl
2243   {
2244     \bool_set_true:N \l_@@_X_columns_aux_bool
2245     \dim_set:Nn \l_@@_X_columns_dim
2246     {

```

The flag `\g_@@_V_of_X_bool` is raised when there is at least in the tabular a column of type `X` using the key `V`. In that case, the width of the `X` column may be considered as correct even though the tabular has not (of course) a width equal to `\l_@@_width_dim`

```

2247       \bool_lazy_and:nnTF
2248       { \g_@@_V_of_X_bool }
2249       { \l_@@_X_columns_aux_bool }
2250       { \dim_use:N \l_@@_X_columns_dim }
2251       {
2252         \dim_compare:nNnTF
2253         {
2254           \dim_abs:n
2255           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2256         }
2257         <
2258         { 0.001 pt }
2259         { \dim_use:N \l_@@_X_columns_dim }
2260         {
2261           \dim_eval:n
2262           {
2263             \l_@@_X_columns_dim
2264             +
2265             \fp_to_dim:n
2266             {
2267               (
2268                 \dim_eval:n
2269                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2270               )
2271               / \fp_use:N \g_@@_total_X_weight_fp
2272             }
2273           }
2274         }
2275       }
2276     }
2277   }
2278 }

```

## 11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

The preamble given by the final user is stored in `\g_@@_user_preamble_tl`. The modified version will be stored in `\g_@@_array_preamble_tl`.

```

2279 \cs_new_protected:Npn \@@_transform_preamble:
2280 {
2281   \@@_transform_preamble_i:
2282   \@@_transform_preamble_ii:
2283 }
2284 \cs_new_protected:Npn \@@_transform_preamble_i:
2285 {
2286   \int_gzero:N \c@jCol

```

The sequence `\g_@@_cols_vlism_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlism`).

```
2287 \seq_gclear:N \g_@@_cols_vlism_seq
```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble provided by the final user.

```
2288 \bool_gset_false:N \g_tmpb_bool
```

The following sequence will store the arguments of the successive `>` in the preamble.

```
2289 \tl_gclear_new:N \g_@@_pre_cell_tl
```

The counter `\l_tmpa_int` will count the number of consecutive occurrences of the symbol `|`.

```
2290 \int_zero:N \l_tmpa_int
2291 \tl_gclear:N \g_@@_array_preamble_tl
2292 \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2293 {
2294   \tl_gset:Nn \g_@@_array_preamble_tl
2295     { ! { \skip_horizontal:N \arrayrulewidth } }
2296 }
2297 {
2298   \clist_if_in:NnT \l_@@_vlines_clist 1
2299   {
2300     \tl_gset:Nn \g_@@_array_preamble_tl
2301       { ! { \skip_horizontal:N \arrayrulewidth } }
2302   }
2303 }
```

Now, we actually make the preamble (which will be given to `{array}`). It will be stored in `\g_@@_array_preamble_tl`.

```
2304 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2305 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2306 \@@_replace_columncolor:
2307 }
```

```
2308 \cs_new_protected:Npn \@@_transform_preamble_ii:
2309 {
```

If there were delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```
2310 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2311 {
2312   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2313   { \bool_gset_true:N \g_@@_delims_bool }
2314 }
2315 { \bool_gset_true:N \g_@@_delims_bool }
```

We want to remind whether there is a specifier `|` at the end of the preamble.

```
2316 \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
```

We complete the preamble with the potential “exterior columns” (on both sides).

```
2317 \int_if_zero:nTF { \l_@@_first_col_int }
2318 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2319 {
2320   \bool_if:NF \g_@@_delims_bool
2321   {
2322     \bool_if:NF \l_@@_tabular_bool
2323     {
2324       \clist_if_empty:NT \l_@@_vlines_clist
2325       {
2326         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2327         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
```



```

2328     }
2329   }
2330 }
2331 }
2332 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2333 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2334 {
2335   \bool_if:NF \g_@@_delims_bool
2336   {
2337     \bool_if:NF \l_@@_tabular_bool
2338     {
2339       \clist_if_empty:NT \l_@@_vlines_clist
2340       {
2341         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2342         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2343       }
2344     }
2345   }
2346 }

```

We try to give a good error message when the final user puts more columns than allowed by the preamble of the array. The mechanism consists of an extra column. However, if tagging is in force, that dummy extra column will be tagged (with <TD> tags) and that's why we disable that mechanism when tagging is in force.

```

2347   \tag_if_active:F
2348   {

```

Moreover, when {NiceTabular\*} is used, the mechanism can't be used for technical reasons. We test that situation with \l\_@@\_tabular\_width\_dim.

```

2349   \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2350   {
2351     \tl_gput_right:Nn \g_@@_array_preamble_tl
2352     { > { \@@_err_too_many_cols: } 1 }
2353   }
2354 }
2355 }

```

We have used to add a last column to raise a good error message when the user puts more columns than allowed by its preamble. For technical reasons, it was not possible to do that in {NiceTabular\*} and that's why we used to control that with the value of \l\_@@\_tabular\_width\_dim).

The preamble provided by the final user will be read by a finite automata. The following function \@@\_rec\_preamble:n will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```

2356 \cs_new_protected:Npn \@@_rec_preamble:n #1
2357 {

```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism \csname...\endcsname. Be careful: all these functions take in as first argument the letter (or token) itself.<sup>11</sup>

```

2358   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2359   { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2360   {

```

Now, the columns defined by \newcolumntype of array.

```

2361   \cs_if_exist:cTF { NC @ find @ #1 }
2362   {
2363     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2364     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2365   }
2366   {

```

---

<sup>11</sup>We do that because it's an easy way to insert the letter at some places in the code that we will add to \g\_@@\_array\_preamble\_tl.

```

2367         \str_if_eq:nnTF { #1 } { S }
2368         { \@@_fatal:n { unknown~column~type~S } }
2369         { \@@_fatal:nn { unknown~column~type } { #1 } }
2370     }
2371 }
2372 }

```

For c, l and r

```

2373 \cs_new_protected:Npn \@@_c: #1
2374 {
2375     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2376     \tl_gclear:N \g_@@_pre_cell_tl
2377     \tl_gput_right:Nn \g_@@_array_preamble_tl
2378     { > \@@_cell_begin: c < \@@_cell_end: }

```

We increment the counter of columns and then we test for the presence of a <.

```

2379     \int_gincr:N \c@jCol
2380     \@@_rec_preamble_after_col:n
2381 }

2382 \cs_new_protected:Npn \@@_l: #1
2383 {
2384     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2385     \tl_gclear:N \g_@@_pre_cell_tl
2386     \tl_gput_right:Nn \g_@@_array_preamble_tl
2387     {
2388         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2389         l
2390         < \@@_cell_end:
2391     }
2392     \int_gincr:N \c@jCol
2393     \@@_rec_preamble_after_col:n
2394 }

2395 \cs_new_protected:Npn \@@_r: #1
2396 {
2397     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2398     \tl_gclear:N \g_@@_pre_cell_tl
2399     \tl_gput_right:Nn \g_@@_array_preamble_tl
2400     {
2401         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2402         r
2403         < \@@_cell_end:
2404     }
2405     \int_gincr:N \c@jCol
2406     \@@_rec_preamble_after_col:n
2407 }

```

For ! and @

```

2408 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2409 {
2410     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2411     \@@_rec_preamble:n
2412 }
2413 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2414 \cs_new_protected:cpn { @@ _ | : } #1
2415 {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

2416     \int_incr:N \l_tmpa_int
2417     \@@_make_preamble_i_i:n
2418 }

```

```

2419 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2420 {

```

Here, we can't use `\str_if_eq:eeTF`.

```

2421 \str_if_eq:nnTF { #1 } { | }
2422 { \use:c { @@ _ | : } | }
2423 { \@@_make_preamble_i_ii:nn { } #1 }
2424 }

```

The following constructions aims to allow cumulative blocks of options between square brackets such as in `[color=blue][tikz=dashed]`.

```

2425 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2426 {
2427 \str_if_eq:nnTF { #2 } { [ ]
2428 { \@@_make_preamble_i_ii:nw { #1 } [ ]
2429 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2430 }
2431 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2432 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2433 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2434 {
2435 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2436 \tl_gput_right:Ne \g_@@_array_preamble_tl
2437 {

```

Here, the command `\dim_use:N` is mandatory.

```

2438 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2439 }
2440 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2441 {
2442 \@@_vline:n
2443 {
2444 position = \int_eval:n { \c_jCol + 1 } ,
2445 multiplicity = \int_use:N \l_tmpa_int ,
2446 total-width = \dim_use:N \l_@@_rule_width_dim ,
2447 #2
2448 }

```

We don't have provided value for `start` nor for `end`, which means that the rule will cover (potentially) all the rows of the array.

```

2449 }
2450 \int_zero:N \l_tmpa_int
2451 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2452 \@@_rec_preamble:n #1
2453 }

```

```

2454 \cs_new_protected:cpn { @@ _ > : } #1 #2
2455 {
2456 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2457 \@@_rec_preamble:n
2458 }
2459 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier `p` (and also the specifiers `m`, `b`, `V` and `X`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys.

```

2460 \keys_define:nn { nicematrix / p-column }
2461 {
2462 r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2463 r .value_forbidden:n = true ,
2464 c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2465 c .value_forbidden:n = true ,
2466 l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,

```

```

2467 l .value_forbidden:n = true ,
2468 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2469 S .value_forbidden:n = true ,
2470 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2471 p .value_forbidden:n = true ,
2472 t .meta:n = p ,
2473 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2474 m .value_forbidden:n = true ,
2475 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2476 b .value_forbidden:n = true
2477 }

```

For p but also b and m.

```

2478 \cs_new_protected:Npn \@@_p: #1
2479 {
2480   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

Now, you look for a potential character [ after the letter of the specifier (for the options).

```

2481   \@@_make_preamble_ii_i:n
2482 }
2483 \cs_new_eq:NN \@@_b: \@@_p:
2484 \cs_new_eq:NN \@@_m: \@@_p:
2485 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2486 {
2487   \str_if_eq:nnTF { #1 } { [ ]
2488     { \@@_make_preamble_ii_ii:w [ ]
2489       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2490     }
2491   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2492     { \@@_make_preamble_ii_iii:nn { #1 } }

```

#1 is the optional argument of the specifier (a list of *key-value* pairs).

#2 is the mandatory argument of the specifier: the width of the column.

```

2493 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2494 {

```

The possible values of \l\_@@\_hpos\_col\_str are j (for *justified* which is the initial value), l, c, r, L, C and R (when the user has used the corresponding key in the optional argument of the specifier).

```

2495   \str_set:Nn \l_@@_hpos_col_str { j }
2496   \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2497   \setlength { \l_tmpa_dim } { #2 }
2498   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2499 }
2500 \cs_new_protected:Npn \@@_keys_p_column:n #1
2501 { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`. The third is some code added at the beginning of the cell.

```

2502 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2503 {

```

Here, `\expanded` would probably be slightly faster than `\use:e`

```

2504   \use:e
2505   {
2506     \@@_make_preamble_ii_vi:nnnnnnnn
2507     { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }

```

```

2508         { #1 }
2509     {
2510         \cs_set_eq:NN \rotate \@@_rotate_p_col:

```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_tl` which will provide the horizontal alignment of the column to which belongs the cell.

```

2511         \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2512         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2513     {

```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```

2514         \def \exp_not:N \l_@@_hpos_cell_tl
2515         { \str_lowercase:f { \l_@@_hpos_col_str } }
2516     }
2517     \IfPackageLoadedTF { ragged2e }
2518     {
2519         \str_case:on \l_@@_hpos_col_str
2520         {

```

The following `\exp_not:N` are mandatory.

```

2521             c { \exp_not:N \Centering }
2522             l { \exp_not:N \RaggedRight }
2523             r { \exp_not:N \RaggedLeft }
2524         }
2525     }
2526     {
2527         \str_case:on \l_@@_hpos_col_str
2528         {
2529             c { \exp_not:N \centering }
2530             l { \exp_not:N \raggedright }
2531             r { \exp_not:N \raggedleft }
2532         }
2533     }
2534     #3
2535 }
2536 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2537 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2538 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2539 { #2 }
2540 {
2541     \str_case:onF \l_@@_hpos_col_str
2542     {
2543         { j } { c }
2544         { si } { c }
2545     }

```

We use `\str_lowercase:n` to convert R to r, etc.

```

2546         { \str_lowercase:f \l_@@_hpos_col_str }
2547     }
2548 }

```

We increment the counter of columns, and then we test for the presence of a `<`.

```

2549     \int_gincr:N \c@jCol
2550     \@@_rec_preamble_after_col:n
2551 }

```

**#1** is the optional argument of `{minipage}` (or `{varwidth}`): t or b. Indeed, for the columns of type m, we use the value b here because there is a special post-action in order to center vertically the box (see **#4**).

**#2** is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

**#3** is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that **#3** some code to fix the value of `\l_@@_hpos_cell_tl` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a `m` column) or nothing (in the other cases).

#5 is a code put just before the `c` (or `r` or `l`: see #8).

#6 is a code put just after the `c` (or `r` or `l`: see #8).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the letter `c` or `r` or `l` which is the basic specifier of column which is used *in fine*.

```

2552 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2553 {
2554   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2555   {
2556     \tl_gput_right:Nn \g_@@_array_preamble_tl
2557     { > \@@_test_if_empty_for_S: }
2558   }
2559   {
2560     \str_if_eq:eeTF { #7 } { varwidth }
2561     {
2562       \tl_gput_right:Nn \g_@@_array_preamble_tl
2563       { > \@@_test_if_empty_varwidth: }
2564     }
2565     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2566   }
2567   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2568   \tl_gclear:N \g_@@_pre_cell_tl
2569   \tl_gput_right:Nn \g_@@_array_preamble_tl
2570   {
2571     > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2572       \dim_set:Nn \l_@@_col_width_dim { #2 }
2573       \@@_cell_begin:

```

We use the form `\minipage–\endminipage` (`\varwidth–\endvarwidth`) for compatibility with `colcell` (2023-10-31).

```

2574       \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty`.

```

2575       \everypar
2576       {
2577         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2578         \everypar { }
2579       }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\RaggedRight`, etc.).

```

2580       #3

```

The following code is to allow something like `\centering` in `\RowStyle`.

```

2581       \g_@@_row_style_tl
2582       \arraybackslash
2583       #5
2584     }
2585     #8
2586     < {
2587       #6

```

The following line has been taken from `array.sty`.

```

2588       \@finalstrut \@arstrutbox
2589       \use:c { end #7 }

```

If the letter in the preamble is `m`, #4 will be equal to `\@@_center_cell_box:` (see just below).

```

2590       #4
2591       \@@_cell_end:
2592     }
2593   }
2594 }

```

The cell always begins with `\ignorespaces` with `array` and that's why we retrieve that token.

```
2595 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2596 {
```

We open a special group with `\group_align_safe_begin:`. Thus, when `\peek_meaning:NTF` will read the `&` (when the cell is empty), that lecture won't trigger the end of the cell (since we are in a lower group...). If the end of cell was triggered, we would have other tokens in the TeX flow (and not `&`).

```
2597 \group_align_safe_begin:
2598 \peek_meaning:NTF &
2599 { \@@_the_cell_is_empty: }
2600 {
2601   \peek_meaning:NTF \\\
2602   { \@@_the_cell_is_empty: }
2603   {
2604     \peek_meaning:NTF \crcr
2605     \@@_the_cell_is_empty:
2606     \group_align_safe_end:
2607   }
2608 }
2609 }
```

A special version of the previous function for the columns of type V (of `varwidth`).

```
2610 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2611 {
2612   \group_align_safe_begin:
2613   \peek_meaning:NTF &
2614   { \@@_the_cell_is_empty_varwidth: }
2615   {
2616     \peek_meaning:NTF \\\
2617     { \@@_the_cell_is_empty_varwidth: }
2618     {
2619       \peek_meaning:NTF \crcr
2620       \@@_the_cell_is_empty_varwidth:
2621       \group_align_safe_end:
2622     }
2623   }
2624 }

2625 \cs_new_protected:Npn \@@_the_cell_is_empty:
2626 {
2627   \group_align_safe_end:
2628   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2629   {
```

Be careful: here, we can't merely use `\bool_gset_true: \g_@@_empty_cell_bool`, in particular because of the columns of type X.

```
2630   \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
```

If all the cells of the column are empty, we still must have a column with the width required by the column of type p (or b, or m).

```
2631   \skip_horizontal:N \l_@@_col_width_dim
2632 }
2633 }

2634 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2635 {
2636   \group_align_safe_end:
2637   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2638   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2639 }
```

```

2640 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2641 {
2642   \peek_meaning:NT \__siunitx_table_skip:n
2643   { \bool_gset_true:N \g_@@_empty_cell_bool }
2644 }

```

The following command will be used in `m-columns` in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more than the height of `\strutbox`, there is only one row.

```

2645 \cs_new_protected:Npn \@@_center_cell_box:
2646 {

```

By putting instructions in `\g_@@_cell_after_hook_tl`, we require a post-action of the box `\l_@@_cell_box`.

```

2647   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2648   {
2649     \dim_compare:nNnT
2650     { \box_ht:N \l_@@_cell_box }
2651     >

```

Previously, we had `\@arstrutbox` and not `\strutbox` in the following line but the code in `array` has changed in v 2.5g and we follow the change (see *array: Correctly identify single-line m-cells* in LaTeX News 36).

```

2652     { \box_ht:N \strutbox }
2653     {
2654       \hbox_set:Nn \l_@@_cell_box
2655       {
2656         \box_move_down:nn
2657         {
2658           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2659             + \baselineskip ) / 2
2660         }
2661         { \box_use:N \l_@@_cell_box }
2662       }
2663     }
2664   }
2665 }

```

For `V` (similar to the `V` of `varwidth`).

```

2666 \cs_new_protected:Npn \@@_V: #1 #2
2667 {
2668   \str_if_eq:nnTF { #2 } { [ ] }
2669   { \@@_make_preamble_V_i:w [ ] }
2670   { \@@_make_preamble_V_i:w [ ] { #2 } }
2671 }
2672 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2673 { \@@_make_preamble_V_ii:nn { #1 } }
2674 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2675 {
2676   \str_set:Nn \l_@@_vpos_col_str { p }
2677   \str_set:Nn \l_@@_hpos_col_str { j }
2678   \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2679   \setlength { \l_tmpa_dim } { #2 }
2680   \IfPackageLoadedTF { varwidth }
2681   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2682   {

```



```

2683 \@@_error_or_warning:n { varwidth-not-loaded }
2684 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2685 }
2686 }

```

For `w` and `W`

```

2687 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2688 \cs_new_protected:Npn \@@_W: { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

**#1** is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;

**#2** is the type of column (`w` or `W`);

**#3** is the type of horizontal alignment (`c`, `l`, `r` or `s`);

**#4** is the width of the column.

```

2689 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2690 {
2691   \str_if_eq:nnTF { #3 } { s }
2692     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2693     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2694 }

```

First, the case of an horizontal alignment equal to `s` (for *stretch*).

**#1** is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;

**#2** is the width of the column.

```

2695 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2696 {
2697   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2698   \tl_gclear:N \g_@@_pre_cell_tl
2699   \tl_gput_right:Nn \g_@@_array_preamble_tl
2700     {
2701       > {

```

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2702   \setlength { \l_@@_col_width_dim } { #2 }
2703   \@@_cell_begin:
2704   \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2705 }
2706 c
2707 < {
2708   \@@_cell_end_for_w_s:
2709   #1
2710   \@@_adjust_size_box:
2711   \box_use_drop:N \l_@@_cell_box
2712 }
2713 }
2714 \int_gincr:N \c@jCol
2715 \@@_rec_preamble_after_col:n
2716 }

```

Then, the most important version, for the horizontal alignments types of `c`, `l` and `r` (and not `s`).

```

2717 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2718 {
2719   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2720   \tl_gclear:N \g_@@_pre_cell_tl
2721   \tl_gput_right:Nn \g_@@_array_preamble_tl
2722     {
2723       > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2724         \setlength { \l_@@_col_width_dim } { #4 }
2725         \hbox_set:Nw \l_@@_cell_box
2726         \@@_cell_begin:
2727         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2728     }
2729     c
2730     < {
2731         \@@_cell_end:
2732         \hbox_set_end:
2733         #1
2734         \@@_adjust_size_box:
2735         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2736     }
2737 }
```

We increment the counter of columns and then we test for the presence of a `<`.

```

2738     \int_gincr:N \c@jCol
2739     \@@_rec_preamble_after_col:n
2740 }

2741 \cs_new_protected:Npn \@@_special_W:
2742 {
2743     \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2744     { \@@_warning:n { W~warning } }
2745 }
```

For `S` (of `siunitx`).

```

2746 \cs_new_protected:Npn \@@_S: #1 #2
2747 {
2748     \str_if_eq:nnTF { #2 } { [ ] }
2749     { \@@_make_preamble_S:w [ ] }
2750     { \@@_make_preamble_S:w [ ] { #2 } }
2751 }

2752 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2753 { \@@_make_preamble_S_i:n { #1 } }

2754 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2755 {
2756     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2757     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2758     \tl_gclear:N \g_@@_pre_cell_tl
2759     \tl_gput_right:Nn \g_@@_array_preamble_tl
2760     {
2761         > {
```

In the cells of a column of type `S`, we have to wrap the command `\@@_node_cell:` for the horizontal alignment of the content of the cell (`siunitx` has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignment once again).

```

2762         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2763         \keys_set:nn { siunitx } { #1 }
2764         \@@_cell_begin:
2765         \siunitx_cell_begin:w
2766     }
2767     c
2768     <
2769     {
2770         \siunitx_cell_end:
```

We want the value of `\l__siunitx_table_text_bool` available *after* `\@@_cell_end:` because we need it to know how to align our box after the construction of the PGF/TikZ node. That's why we use `\g_@@_cell_after_hook_tl` to reset the correct value of `\l__siunitx_table_text_bool` (of course, it will stay local within the cell of the underlying `\halign`).

```

2771         \tl_gput_right:Nc \g_@@_cell_after_hook_tl
2772         {
2773             \bool_if:NTF \l__siunitx_table_text_bool
2774             { \bool_set_true:N }
2775             { \bool_set_false:N }
2776             \l__siunitx_table_text_bool
2777         }
2778     \@@_cell_end:
2779 }
2780 }

```

We increment the counter of columns and then we test for the presence of a `<`.

```

2781     \int_gincr:N \c@jCol
2782     \@@_rec_preamble_after_col:n
2783 }

```

For `(`, `[` and `\{`.

```

2784 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2785 {
2786     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in `{NiceArray}`, we reserve space for the left delimiter.

```

2787     \int_if_zero:nTF { \c@jCol }
2788     {
2789         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2790         {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

2791         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2792         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2793         \@@_rec_preamble:n #2
2794     }
2795     {
2796         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2797         \@@_make_preamble_iv:nn { #1 } { #2 }
2798     }
2799 }
2800 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2801 }
2802 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2803 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2804 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2805 {
2806     \tl_gput_right:Nc \g_@@_pre_code_after_tl
2807     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2808     \tl_if_in:nnTF { ( [ \{ ) } \} \left \right } { #2 }
2809     {
2810         \@@_error:nn { delimiter~after~opening } { #2 }
2811         \@@_rec_preamble:n
2812     }
2813     { \@@_rec_preamble:n #2 }
2814 }

```

In fact, it would be possible to define `\left` and `\right` as no-op.

```

2815 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2816 { \use:c { @@ _ \token_to_str:N ( : } }

```

For the closing delimiters. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have an opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is `{NiceArray}`).

```

2817 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2818 {
2819   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2820   \tl_if_in:nnTF { ) ] \} } { #2 }
2821   { \@@_make_preamble_v:nnn #1 #2 }
2822   {
2823     \str_if_eq:nnTF { \s_stop } { #2 }
2824     {
2825       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2826       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2827       {
2828         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2829         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2830         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2831         \@@_rec_preamble:n #2
2832       }
2833     }
2834     {
2835       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2836       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2837       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2838       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2839       \@@_rec_preamble:n #2
2840     }
2841   }
2842 }
2843 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2844 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2845 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2846 {
2847   \str_if_eq:nnTF { \s_stop } { #3 }
2848   {
2849     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2850     {
2851       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2852       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2853       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2854       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2855     }
2856     {
2857       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2858       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2859       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2860       \@@_error:nn { double~closing~delimiter } { #2 }
2861     }
2862   }
2863   {
2864     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2865     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2866     \@@_error:nn { double~closing~delimiter } { #2 }
2867     \@@_rec_preamble:n #3
2868   }
2869 }
2870 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2871 { \use:c { @@ _ \token_to_str:N ) : } }

```

After a specifier of column, we have to test whether there is one or several `<{. .}` because, after those

potential <{...}, we have to insert !{\skip\_horizontal:N ...} when the key vlines is used. In fact, we have also to test whether there is, after the <{...}, a @{...}.

```

2872 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2873 {
2874   \str_if_eq:nnTF { #1 } { < }
2875   { \@@_rec_preamble_after_col_i:n }
2876   {
2877     \str_if_eq:nnTF { #1 } { @ }
2878     { \@@_rec_preamble_after_col_ii:n }
2879     {
2880       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2881       {
2882         \tl_gput_right:Nn \g_@@_array_preamble_tl
2883         { ! { \skip_horizontal:N \arrayrulewidth } }
2884       }
2885       {
2886         \clist_if_in:NcT \l_@@_vlines_clist
2887         { \int_eval:n { \c@jCol + 1 } }
2888         {
2889           \tl_gput_right:Nn \g_@@_array_preamble_tl
2890           { ! { \skip_horizontal:N \arrayrulewidth } }
2891         }
2892       }
2893       \@@_rec_preamble:n { #1 }
2894     }
2895   }
2896 }
2897 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2898 {
2899   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2900   \@@_rec_preamble_after_col:n
2901 }

```

We have to catch a @{...} after a specifier of column because, if we have to draw a vertical rule, we have to add in that @{...} a \hskip corresponding to the width of the vertical rule.

```

2902 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2903 {
2904   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2905   {
2906     \tl_gput_right:Nn \g_@@_array_preamble_tl
2907     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2908   }
2909   {
2910     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2911     {
2912       \tl_gput_right:Nn \g_@@_array_preamble_tl
2913       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2914     }
2915     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2916   }
2917   \@@_rec_preamble:n
2918 }
2919 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2920 {
2921   \tl_clear:N \l_tmpa_tl
2922   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2923   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2924 }

```

The token \NC@find is at the head of the definition of the columns type done by \newcolumnstype. We want that token to be no-op here.

```

2925 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2926 { \@@_rec_preamble:n }

```

For the case of a letter X. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a [ after the letter X.

```

2927 \cs_new_protected:Npn \@@_X: #1 #2
2928 {
2929   \str_if_eq:nnTF { #2 } { [ ]
2930     { \@@_make_preamble_X:w [ ]
2931       { \@@_make_preamble_X:w [ ] #2 }
2932     }
2933   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2934   { \@@_make_preamble_X:i:n { #1 } }

```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { `nicematrix / p-column` } but also a key V and also a key which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```

2935 \keys_define:nn { nicematrix / X-column }
2936 {
2937   V .code:n =
2938   \IfPackageLoadedTF { varwidth }
2939   {
2940     \bool_set_true:N \l_@@_V_of_X_bool
2941     \bool_gset_true:N \g_@@_V_of_X_bool
2942   }
2943   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2944   unknown .code:n =
2945   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2946   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2947   { \@@_error_or_warning:n { invalid~weight } }
2948 }

```

In the following command, #1 is the list of the options of the specifier X.

```

2949 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2950 {

```

The possible values of `\l_@@_hpos_col_str` are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier X).

```

2951   \str_set:Nn \l_@@_hpos_col_str { j }

```

The possible values of `\l_@@_vpos_col_str` are p (the initial value), m and b (when the user has used the corresponding key in the optional argument of the specifier X).

```

2952   \str_set:Nn \l_@@_vpos_col_str { p }

```

We will store in `\l_tmpa_fp` the weight of the column (`\l_tmpa_fp` also appears in {`nicematrix/X-column`} and the error message `invalid~weight`).

```

2953   \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

2954   \@@_keys_p_column:n { #1 }

```

The unknown keys have been stored by `\@@_keys_p_column:n` in `\l_tmpa_tl` and we use them right away in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitly provided by the final user.

```

2955   \bool_set_false:N \l_@@_V_of_X_bool
2956   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

Now, the weight of the column is stored in `\l_tmpa_tl`.

```

2957   \fp_gadd:Nn \g_@@_total_X_weight_fp \l_tmpa_fp

```

We test whether we know the actual width of the X-columns by reading the `aux` file (after the first compilation, the width of the X-columns is computed and written in the `aux` file).

```
2958 \bool_if:NTF \l_@@_X_columns_aux_bool
2959 {
2960 \@@_make_preamble_ii_iv:nnn
```

Of course, the weight of a column depends of its weight (in `\l_tmpa_fp`).

```
2961 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2962 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2963 { \@@_no_update_width: }
2964 }
```

In the current compilation, we don't know the actual width of the X column. However, you have to construct the cells of that column! By convention, we have decided to compose in a `{minipage}` of width 5 cm even though we will nullify `\l_@@_cell_box` after its composition.

```
2965 {
2966 \tl_gput_right:Nn \g_@@_array_preamble_tl
2967 {
2968 > {
2969 \@@_cell_begin:
2970 \bool_set_true:N \l_@@_X_bool
```

You encounter a problem on 2023-03-04: for an environment with X columns, during the first compilations (which are not the definitive one), sometimes, some cells are declared empty even if they should not. That's a problem because user's instructions may use these nodes. That's why we have added the following `\NotEmpty`.

```
2971 \NotEmpty
```

The following code will nullify the box of the cell.

```
2972 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2973 { \hbox_set:Nn \l_@@_cell_box { } }
```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```
2974 \begin { minipage } { 5 cm } \arraybackslash
2975 }
2976 c
2977 < {
2978 \end { minipage }
2979 \@@_cell_end:
2980 }
2981 }
2982 \int_gincr:N \c@jCol
2983 \@@_rec_preamble_after_col:n
2984 }
2985 }
```

```
2986 \cs_new_protected:Npn \@@_no_update_width:
2987 {
2988 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2989 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2990 }
```

For the letter set by the user with `vlines-in-sub-matrix` (`vlism`).

```
2991 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2992 {
2993 \seq_gput_right:Ne \g_@@_cols_vlism_seq
2994 { \int_eval:n { \c@jCol + 1 } }
2995 \tl_gput_right:Ne \g_@@_array_preamble_tl
2996 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2997 \@@_rec_preamble:n
2998 }
```

The token `\s_stop` is a marker that we have inserted to mark the end of the preamble (as provided by the final user) that we have inserted in the TeX flow.

```
2999 \cs_set_eq:cn { @@ _ \token_to_str:N \s_stop : } \use_none:n
```

The following lines try to catch some errors (when the final user has forgotten the preamble of its environment).

```
3000 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3001 { \@@_fatal:n { Preamble-forgotten } }
3002 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3003 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3004 { @@ _ \token_to_str:N \hline : }
3005 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3006 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3007 { @@ _ \token_to_str:N \hline : }
3008 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3009 { @@ _ \token_to_str:N \hline : }
3010 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3011 { @@ _ \token_to_str:N \hline : }
3012 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3013 { @@ _ \token_to_str:N \hline : }
```

## 12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```
3014 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3015 {
```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```
3016 \multispan { #1 }
3017 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3018 \begingroup
3019 \tbl_update_multicolumn_cell_data:n { #1 }
```

Now, we patch the (small) preamble as we have done with the main preamble of the array.

```
3020 \tl_gclear:N \g_@@_preamble_tl
3021 \@@_make_m_preamble:n #2 \q_stop
```

The following lines are an adaptation of the definition of `\multicolumn` in `array`.

```
3022 \def \@addamp
3023 {
3024 \legacy_if:nTF { @firstamp }
3025 { \legacy_if_set_false:n { @firstamp } }
3026 { \@preamerr 5 }
3027 }
3028 \exp_args:No \mkpream \g_@@_preamble_tl
3029 \@addtopreamble \empty
3030 \endgroup
3031 \UseTaggingSocket { tbl / colspan } { #1 }
```

Now, we do a treatment specific to `nicematrix` which has no equivalent in the original definition of `\multicolumn`.

```
3032 \int_compare:nNnT { #1 } > { \c_one_int }
3033 {
3034 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
3035 { \int_use:N \c_iRow - \int_eval:n { \c_jCol + 1 } }
3036 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
```



```

3037 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3038 {
3039   {
3040     \int_if_zero:nTF { \c@jCol }
3041       { \int_eval:n { \c@iRow + 1 } }
3042       { \int_use:N \c@iRow }
3043   }
3044   { \int_eval:n { \c@jCol + 1 } }
3045   {
3046     \int_if_zero:nTF { \c@jCol }
3047       { \int_eval:n { \c@iRow + 1 } }
3048       { \int_use:N \c@iRow }
3049   }
3050   { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3051   { }
3052 }
3053 }

```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of `colortbl` is available in `\multicolumn`.

```

3054 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3055 {
3056   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3057   {
3058     \@@_rectanglecolor [ ##1 ]
3059     { \exp_not:n { ##2 } }
3060     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3061     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3062   }
3063   \ignorespaces
3064 }

```

The following lines were in the original definition of `\multicolumn`.

```

3065 \def \@sharp { #3 }
3066 \@arstrut
3067 \@preamble
3068 \null

```

We add some lines.

```

3069 \int_gadd:Nn \c@jCol { #1 - 1 }
3070 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
3071   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3072 \ignorespaces
3073 }

```

The following commands will patch the (small) preamble of the `\multicolumn`. All those commands have a `m` in their name to recall that they deal with the redefinition of `\multicolumn`.

```

3074 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3075 {
3076   \str_case:nnF { #1 }
3077   {
3078     c { \@@_make_m_preamble_i:n #1 }
3079     l { \@@_make_m_preamble_i:n #1 }
3080     r { \@@_make_m_preamble_i:n #1 }
3081     > { \@@_make_m_preamble_ii:nn #1 }
3082     ! { \@@_make_m_preamble_ii:nn #1 }
3083     @ { \@@_make_m_preamble_ii:nn #1 }
3084     | { \@@_make_m_preamble_iii:n #1 }
3085     p { \@@_make_m_preamble_iv:nnn t #1 }
3086     m { \@@_make_m_preamble_iv:nnn c #1 }

```

```

3087     b { \@@_make_m_preamble_iv:nnn b #1 }
3088     w { \@@_make_m_preamble_v:nnnn { } #1 }
3089     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3090     \q_stop { }
3091   }
3092   {
3093     \cs_if_exist:cTF { NC @ find @ #1 }
3094     {
3095       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3096       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3097     }
3098     {
3099       \str_if_eq:nnTF { #1 } { S }
3100       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3101       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3102     }
3103   }
3104 }

```

For c, l and r

```

3105 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3106 {
3107   \tl_gput_right:Nn \g_@@_preamble_tl
3108   {
3109     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3110     #1
3111     < \@@_cell_end:
3112   }

```

We test for the presence of a <.

```

3113   \@@_make_m_preamble_x:n
3114 }

```

For >, ! and @

```

3115 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3116 {
3117   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3118   \@@_make_m_preamble:n
3119 }

```

For |

```

3120 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3121 {
3122   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3123   \@@_make_m_preamble:n
3124 }

```

For p, m and b

```

3125 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3126 {
3127   \tl_gput_right:Nn \g_@@_preamble_tl
3128   {
3129     > {
3130       \@@_cell_begin:

```

We use `\setlength` instead of `\dim_set:N` to allow a specifier like `p{\widthof{Some words}}`. `widthof` is a command provided by `calc`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

3131     \setlength { \l_tmpa_dim } { #3 }
3132     \begin { minipage } [ #1 ] { \l_tmpa_dim }
3133     \mode_leave_vertical:
3134     \arraybackslash
3135     \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim

```

```

3136     }
3137     c
3138     < {
3139         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3140         \end { minipage }
3141         \@@_cell_end:
3142     }
3143 }

```

We test for the presence of a <.

```

3144 \@@_make_m_preamble_x:n
3145 }

```

For w and W

```

3146 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3147 {
3148     \tl_gput_right:Nn \g_@@_preamble_tl
3149     {
3150         > {
3151             \dim_set:Nn \l_@@_col_width_dim { #4 }
3152             \hbox_set:Nw \l_@@_cell_box
3153             \@@_cell_begin:
3154             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3155         }
3156         c
3157         < {
3158             \@@_cell_end:
3159             \hbox_set_end:
3160             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3161             #1
3162             \@@_adjust_size_box:
3163             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3164         }
3165     }
}

```

We test for the presence of a <.

```

3166 \@@_make_m_preamble_x:n
3167 }

```

After a specifier of column, we have to test whether there is one or several <{..}.

```

3168 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3169 {
3170     \str_if_eq:nnTF { #1 } { < }
3171     { \@@_make_m_preamble_ix:n }
3172     { \@@_make_m_preamble:n { #1 } }
3173 }
3174 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3175 {
3176     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3177     \@@_make_m_preamble_x:n
3178 }

```

The command \@@\_put\_box\_in\_flow: puts the box \l\_tmpa\_box (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the depth to take back into account the potential exterior rows (the total height of the first row has been computed in \l\_tmpa\_dim and the total height of the potential last row in \l\_tmpb\_dim).

```

3179 \cs_new_protected:Npn \@@_put_box_in_flow:
3180 {
3181     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3182     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3183     \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3184     { \box_use_drop:N \l_tmpa_box }
3185     { \@@_put_box_in_flow_i: }
3186 }

```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different of `c` (the initial value).

```

3187 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3188 {
3189   \pgfpicture
3190     \@@_qpoint:n { row - 1 }
3191     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3192     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3193     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3194     \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }

```

Now, `\g_tmpa_dim` contains the  $y$ -value of the center of the array (the delimiters are centered in relation with this value).

```

3195   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3196   {
3197     \int_set:Nn \l_tmpa_int
3198     { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3199     \bool_lazy_or:nnT
3200     { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3201     { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3202     {
3203       \@@_error:n { bad-value-for-baseline-line }
3204       \int_set_eq:NN \l_tmpa_int \c_one_int
3205     }
3206     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3207   }
3208   {
3209     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3210     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3211     {
3212       \str_if_eq:onTF \l_@@_baseline_tl { b }
3213       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3214       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3215     }
3216     \bool_lazy_or:nnT
3217     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3218     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3219     {
3220       \@@_error:n { bad-value-for-baseline }
3221       \int_set_eq:NN \l_tmpa_int \c_one_int
3222     }
3223     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3224     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3225   }
3226   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

Now, `\g_tmpa_dim` contains the value of the  $y$  translation we have to do.

```

3227   \endpgfpicture
3228   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3229   \box_use_drop:N \l_tmpa_box
3230 }

```

The following command is *always* used by `{NiceArrayWithDelims}` (even if, in fact, there is no tabular notes: in fact, it's not possible to know whether there is tabular notes or not before the composition of the blocks).

```

3231 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3232 {

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

3233   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3234 {
3235   \int_compare:nNtT { \c@jCol } > { \c_one_int }
3236   {
3237     \box_set_wd:Nn \l_@@_the_array_box
3238     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3239   }
3240 }

```

We need a `{minipage}` because we will insert a LaTeX list for the tabular notes (that means that a `\vtop{\hsize=...}` is not enough).

```

3241 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3242 \bool_if:NT \l_@@_caption_above_bool
3243 {
3244   \tl_if_empty:NF \l_@@_caption_tl
3245   {
3246     \bool_set_false:N \g_@@_caption_finished_bool
3247     \int_gzero:N \c@tabularnote
3248     \@@_insert_caption:

```

If there is one or several commands `\tabularnote` in the caption, we will write in the `aux` file the number of such tabular notes... but only the tabular notes for which the command `\tabularnote` has been used without its optional argument (between square brackets).

```

3249   \int_compare:nNtT { \g_@@_notes_caption_int } > { \c_zero_int }
3250   {
3251     \tl_gput_right:Ne \g_@@_aux_tl
3252     {
3253       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3254       { \int_use:N \g_@@_notes_caption_int }
3255     }
3256     \int_gzero:N \g_@@_notes_caption_int
3257   }
3258 }
3259 }

```

The `\hbox` avoids that the `pgfpicture` inside `\@@_draw_blocks` adds a extra vertical space before the notes.

```

3260 \hbox
3261 {
3262   \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right away because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are `medium` nodes to create for the blocks.

```

3263   \@@_create_extra_nodes:
3264   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3265 }

```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because it compiles twice its tabular).

```

3266 \bool_lazy_any:nT
3267 {
3268   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3269   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3270   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3271 }
3272 \@@_insert_tabularnotes:
3273 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3274 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3275 \end { minipage }
3276 }

```

```

3277 \cs_new_protected:Npn \@@_insert_caption:
3278 {
3279   \tl_if_empty:NF \l_@@_caption_tl
3280   {
3281     \cs_if_exist:NTF \@capttype
3282     { \@@_insert_caption_i: }
3283     { \@@_error:n { caption-outside-float } }
3284   }
3285 }

```

```

3286 \cs_new_protected:Npn \@@_insert_caption_i:
3287 {
3288   \group_begin:

```

The flag `\l_@@_in_caption_bool` affects only the behavior of the command `\tabularnote` when used in the caption.

```

3289   \bool_set_true:N \l_@@_in_caption_bool

```

The package `floatrow` does a redefinition of `\@makecaption` which will extract the caption from the tabular. However, the old version of `\@makecaption` has been stored by `floatrow` in `\FR@makecaption`. That's why we restore the old version.

```

3290   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3291   \tl_if_empty:NTF \l_@@_short_caption_tl
3292   { \caption }
3293   { \caption [ \l_@@_short_caption_tl ] }
3294   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3295   \bool_if:NF \g_@@_caption_finished_bool
3296   {
3297     \bool_gset_true:N \g_@@_caption_finished_bool
3298     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3299     \int_gzero:N \c@tabularnote
3300   }
3301   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3302   \group_end:
3303 }

```

```

3304 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3305 {
3306   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3307   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3308 }

```

```

3309 \cs_new_protected:Npn \@@_insert_tabularnotes:
3310 {
3311   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3312   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3313   \skip_vertical:N 0.65ex

```

The TeX group is for potential specifications in the `\l_@@_notes_code_before_tl`.

```

3314   \group_begin:
3315   \l_@@_notes_code_before_tl
3316   \tl_if_empty:NF \g_@@_tabularnote_tl
3317   {
3318     \g_@@_tabularnote_tl \par
3319     \tl_gclear:N \g_@@_tabularnote_tl
3320   }

```

We compose the tabular notes with a list of `enumitem`. The `\strut` and the `\unskip` are designed to give the ability to put a `\bottomrule` at the end of the notes with a good vertical space.

```

3321 \int_compare:nNtT { \c@tabularnote } > { \c_zero_int }
3322 {
3323   \bool_if:NTF \l_@@_notes_para_bool
3324   {
3325     \begin { tabularnotes* }
3326       \seq_map_inline:Nn \g_@@_notes_seq
3327       { \@@_one_tabularnote:nn ##1 }
3328       \strut
3329     \end { tabularnotes* }

```

The following `\par` is mandatory for the event that the user has put `\footnotesize` (for example) in the `notes/code-before`.

```

3330   \par
3331 }
3332 {
3333   \tabularnotes
3334     \seq_map_inline:Nn \g_@@_notes_seq
3335     { \@@_one_tabularnote:nn ##1 }
3336     \strut
3337   \endtabularnotes
3338 }
3339 }
3340 \unskip
3341 \group_end:
3342 \bool_if:NT \l_@@_notes_bottomrule_bool
3343 {
3344   \IfPackageLoadedTF { booktabs }
3345   {

```

The two dimensions `\aboverulesep` et `\heavyrulewidth` are parameters defined by `booktabs`.

```

3346   \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

3347   { \CT@arc@ \hrule height \heavyrulewidth }
3348 }
3349 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3350 }
3351 \l_@@_notes_code_after_tl
3352 \seq_gclear:N \g_@@_notes_seq
3353 \seq_gclear:N \g_@@_notes_in_caption_seq
3354 \int_gzero:N \c@tabularnote
3355 }

```

The following command will format (after the main tabular) one tabularnote (with the command `\item`). `#1` is the label (when the command `\tabularnote` has been used with an optional argument between square brackets) and `#2` is the text of the note. The second argument is provided by curryfication.

```

3356 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3357 {
3358   \tl_if_novalue:NTF { #1 }
3359   { \item }
3360   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3361 }

```

The case of baseline equal to `b`. Remember that, when the key `b` is used, the `{array}` (of `array`) is constructed with the option `t` (and not `b`). Now, we do the translation to take into account the option `b`.

```

3362 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3363 {
3364   \pgfpicture
3365     \@@_qpoint:n { row - 1 }
3366     \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```

```

3367 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3368 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3369 \endpgfpicture
3370 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3371 \int_if_zero:nT { \l_@@_first_row_int }
3372 {
3373   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3374   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3375 }
3376 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3377 }

```

Now, the general case.

```

3378 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3379 {

```

We convert a value of  $t$  to a value of 1.

```

3380 \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3381 { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

Now, we convert the value of  $\l_@@_baseline\_tl$  (which should represent an integer) to an integer stored in  $\l\_tmpa\_int$ .

```

3382 \pgfpicture
3383 \@@_qpoint:n { row - 1 }
3384 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3385 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3386 {
3387   \int_set:Nn \l_tmpa_int
3388   {
3389     \str_range:Nnn
3390     \l_@@_baseline_tl
3391     { 6 }
3392     { \tl_count:o \l_@@_baseline_tl }
3393   }
3394   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3395 }
3396 {
3397   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3398   \bool_lazy_or:nnT
3399   { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3400   { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3401   {
3402     \@@_error:n { bad~value~for~baseline }
3403     \int_set:Nn \l_tmpa_int 1
3404   }
3405   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3406 }
3407 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3408 \endpgfpicture
3409 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3410 \int_if_zero:nT { \l_@@_first_row_int }
3411 {
3412   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3413   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3414 }
3415 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3416 }

```

The command  $\@@\_put\_box\_in\_flow\_bis:$  is used when the option `delimiters/max-width` is used because, in this case, we have to adjust the widths of the delimiters. The arguments #1 and #2 are the delimiters specified by the user.

```

3417 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3418 {

```



We will compute the real width of both delimiters used.

```

3419 \dim_zero_new:N \l_@@_real_left_delim_dim
3420 \dim_zero_new:N \l_@@_real_right_delim_dim
3421 \hbox_set:Nn \l_tmpb_box
3422 {
3423   \m@th
3424   $ % $
3425   \left #1
3426   \vcenter
3427   {
3428     \vbox_to_ht:nn
3429     { \box_ht_plus_dp:N \l_tmpa_box }
3430     { }
3431   }
3432   \right .
3433   $ % $
3434 }
3435 \dim_set:Nn \l_@@_real_left_delim_dim
3436 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3437 \hbox_set:Nn \l_tmpb_box
3438 {
3439   \m@th
3440   $ % $
3441   \left .
3442   \vbox_to_ht:nn
3443   { \box_ht_plus_dp:N \l_tmpa_box }
3444   { }
3445   \right #2
3446   $ % $
3447 }
3448 \dim_set:Nn \l_@@_real_right_delim_dim
3449 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.

```

3450 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3451 \@@_put_box_in_flow:
3452 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3453 }

```

The construction of the array in the environment `{NiceArrayWithDelims}` is, in fact, done by the environment `{@@-light-syntax}` or by the environment `{@@-normal-syntax}` (whether the option `light-syntax` is in force or not). When the key `light-syntax` is not used, the construction is a standard environment (and, thus, it's possible to use verbatim in the array).

```

3454 \NewDocumentEnvironment { @@-normal-syntax } { }

```

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it's only a security). In order to detect whether it is empty, we test whether the next token is `\end` and, if it's the case, we test if this is the end of the environment (if it is not, a standard error will be raised by LaTeX for incorrect nested environments).

```

3455 {
3456   \peek_remove_spaces:n
3457   {
3458     \peek_meaning:NTF \end
3459     { \@@_analyze_end:Nn }
3460     {
3461       \@@_transform_preamble:

```

Here is the call to `\array` (we have a dedicated macro `\@@_array:n` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3462     \@@_array:o \g_@@_array_preamble_tl
3463   }
3464 }
3465 }

```

```

3466 {
3467   \@@_create_col_nodes:
3468   \endarray
3469 }

```

When the key `light-syntax` is in force, we use an environment which takes its whole body as an argument (with the specifier `b`).

```

3470 \NewDocumentEnvironment { @@-light-syntax } { b }
3471 {

```

First, we test whether the environment is empty. It's only a security. Of course, this test is more easy than the similar test for the "normal syntax" because we have the whole body of the environment in `#1`.

```

3472   \tl_if_empty:nT { #1 }
3473   { \@@_fatal:n { empty-environment } }
3474   \tl_if_in:nnT { #1 } { & }
3475   { \@@_fatal:n { ampersand-in-light-syntax } }
3476   \tl_if_in:nnT { #1 } { \ }
3477   { \@@_fatal:n { double-backslash-in-light-syntax } }

```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

```

3478   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

The command `\array` is hidden somewhere in `\@@_light_syntax_i:w`.

```

3479 }

```

Now, the second part of the environment. We must leave these lines in the second part (and not put them in the first part even though we caught the whole body of the environment with an argument of type `b`) in order to have the columns `S` of `siunitx` working fine.

```

3480 {
3481   \@@_create_col_nodes:
3482   \endarray
3483 }
3484 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3485 {
3486   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

The body of the array, which is stored in the argument `#1`, is now split into items (and *not* tokens).

```

3487   \seq_clear_new:N \l_@@_rows_seq

```

We rescan the character of end of line in order to have the correct catcode.

```

3488   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3489   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3490   { \seq_set_split:Nee }
3491   { \seq_set_split:Non }
3492   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3493   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3494   \tl_if_empty:NF \l_tmpa_tl
3495   { \seq_put_right:No \l_@@_rows_seq \l_tmpa_tl }

```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```

3496   \int_compare:nNt { \l_@@_last_row_int } = { -1 }
3497   { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }

```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
3498 \tl_build_begin:N \l_@@_new_body_tl
3499 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3500 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3501 \@@_line_with_light_syntax:o \l_tmpa_tl
```

Now, the other rows (with the same treatment, excepted that we have to insert `\` between the rows).

```
3502 \seq_map_inline:Nn \l_@@_rows_seq
3503 {
3504   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3505   \@@_line_with_light_syntax:n { #1 }
3506 }
3507 \tl_build_end:N \l_@@_new_body_tl
3508 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3509 {
3510   \int_set:Nn \l_@@_last_col_int
3511     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3512 }
```

Now, we can construct the preamble: if the user has used the key `last-col`, we have the correct number of columns even though the user has used `last-col` without value.

```
3513 \@@_transform_preamble:
```

The call to `\array` is in the following command (we have a dedicated macro `\@@_array`: because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```
3514 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3515 }
3516 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3517 {
3518   \seq_clear_new:N \l_@@_cells_seq
3519   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3520   \int_set:Nn \l_@@_nb_cols_int
3521     {
3522       \int_max:nn
3523         { \l_@@_nb_cols_int }
3524         { \seq_count:N \l_@@_cells_seq }
3525     }
3526   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3527   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3528   \seq_map_inline:Nn \l_@@_cells_seq
3529     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3530 }
3531 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```
3532 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3533 {
3534   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3535   { \@@_fatal:n { empty-environment } }
```

We repeat in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```
3536 \end { #2 }
3537 }
```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns such as `columns-width`).

```

3538 \cs_new:Npn \@@_create_col_nodes:
3539 {
3540   \crrc
3541   \int_if_zero:nT { \l_@@_first_col_int }
3542   {
3543     \omit
3544     \hbox_overlap_left:n
3545     {
3546       \bool_if:NT \l_@@_code_before_bool
3547       { \pgfsys@markposition { \@@_env: - col - 0 } }
3548       \pgfpicture
3549       \pgfrememberpicturepositiononpagetrue
3550       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3551       \str_if_empty:NF \l_@@_name_str
3552       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3553       \endpgfpicture
3554       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3555     }
3556     &
3557   }
3558   \omit

```

The following instruction must be put after the instruction `\omit` since, of course, it is not expandable.

```

3559   \bool_gset_true:N \g_@@_row_of_col_done_bool

```

First, we put a `col` node on the left of the first column (of course, we have to do that *after* the `\omit`).

```

3560   \int_if_zero:nTF { \l_@@_first_col_int }
3561   {
3562     \@@_mark_position:n { 1 }
3563     \pgfpicture
3564     \pgfrememberpicturepositiononpagetrue
3565     \pgfcoordinate { \@@_env: - col - 1 }
3566     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3567     \str_if_empty:NF \l_@@_name_str
3568     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3569     \endpgfpicture
3570   }
3571   {
3572     \bool_if:NT \l_@@_code_before_bool
3573     {
3574       \hbox
3575       {
3576         \skip_horizontal:n { 0.5 \arrayrulewidth }
3577         \pgfsys@markposition { \@@_env: - col - 1 }
3578         \skip_horizontal:n { -0.5 \arrayrulewidth }
3579       }
3580     }
3581     \pgfpicture
3582     \pgfrememberpicturepositiononpagetrue
3583     \pgfcoordinate { \@@_env: - col - 1 }
3584     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3585     \@@_node_alias:n { 1 }
3586     \endpgfpicture
3587   }

```

We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use that variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but we will add some dimensions to it.

```

3588 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3589 \bool_if:NF \l_@@_auto_columns_width_bool
3590 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3591 {
3592   \bool_lazy_and:nnTF
3593     { \l_@@_auto_columns_width_bool }
3594     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3595     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3596     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3597   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3598 }
3599 \skip_horizontal:N \g_tmpa_skip
3600 \hbox
3601 {
3602   \@@_mark_position:n { 2 }
3603   \pgfpicture
3604   \pgfrememberpicturepositiononpagetrue
3605   \pgfcoordinate { \@@_env: - col - 2 }
3606   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3607   \@@_node_alias:n { 2 }
3608   \endpgfpicture
3609 }

```

We begin a loop over the columns. The integer `\g_tmpa_int` will be the number of the current column. This integer is used for the TikZ nodes.

```

3610 \int_gset_eq:NN \g_tmpa_int \c_one_int
3611 \bool_if:NTF \g_@@_last_col_found_bool
3612 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3613 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3614 {
3615   &
3616   \omit
3617   \int_gincr:N \g_tmpa_int

```

The incrementation of the counter `\g_tmpa_int` must be done after the `\omit` of the cell.

```

3618 \skip_horizontal:N \g_tmpa_skip
3619 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

We create the col node on the right of the current column.

```

3620 \pgfpicture
3621 \pgfrememberpicturepositiononpagetrue
3622 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3623 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3624 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3625 \endpgfpicture
3626 }

```

If there is only one column (and a potential “last column”), we don’t have to put the following code (there is only one column and we have put the correct code previously).

```

3627 \bool_lazy_or:nnF
3628 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3629 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3630 {
3631   &
3632   \omit
3633   \skip_horizontal:N \g_tmpa_skip
3634   \int_gincr:N \g_tmpa_int
3635   \bool_lazy_any:nF
3636   {
3637     \g_@@_delims_bool
3638     \l_@@_tabular_bool
3639     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3640     \l_@@_exterior_arraycolsep_bool

```

```

3641         \l_@@_bar_at_end_of_pream_bool
3642     }
3643     { \skip_horizontal:n { - \col@sep } }
3644     \bool_if:NT \l_@@_code_before_bool
3645     {
3646         \hbox
3647         {
3648             \skip_horizontal:n { -0.5 \arrayrulewidth }

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

3649         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3650         { \skip_horizontal:n { - \arraycolsep } }
3651         \pgfsys@markposition
3652         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3653         \skip_horizontal:n { 0.5 \arrayrulewidth }
3654         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3655         { \skip_horizontal:N \arraycolsep }
3656     }
3657 }
3658 \pgfpicture
3659 \pgfrememberpicturepositiononpagetrue
3660 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3661 {
3662     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3663     {
3664         \pgfpoint
3665         { - 0.5 \arrayrulewidth - \arraycolsep }
3666         \c_zero_dim
3667     }
3668     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3669 }
3670 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3671 \endpgfpicture
3672 }

```

```

3673 \bool_if:NT \g_@@_last_col_found_bool
3674 {
3675     \hbox_overlap_right:n
3676     {
3677         \skip_horizontal:N \g_@@_width_last_col_dim
3678         \skip_horizontal:N \col@sep
3679         \bool_if:NT \l_@@_code_before_bool
3680         {
3681             \pgfsys@markposition
3682             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3683         }
3684         \pgfpicture
3685         \pgfrememberpicturepositiononpagetrue
3686         \pgfcoordinate
3687         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3688         \pgfpointorigin
3689         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3690         \endpgfpicture
3691     }
3692 }
3693 }

```

```

3694 \cs_new_protected:Npn \@@_mark_position:n #1
3695 {
3696     \bool_if:NT \l_@@_code_before_bool

```

```

3697     {
3698     \hbox
3699     {
3700         \skip_horizontal:n { -0.5 \arrayrulewidth }
3701         \pgfsys@markposition { \@@_env: - col - #1 }
3702         \skip_horizontal:n { 0.5 \arrayrulewidth }
3703     }
3704 }
3705 }
3706 \cs_new_protected:Npn \@@_node_alias:n #1
3707 {
3708     \str_if_empty:NF \l_@@_name_str
3709     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3710 }

```

Here is the preamble for the “first column” (if the user uses the key `first-col`)

```

3711 \tl_const:Nn \c_@@_preamble_first_col_tl
3712 {
3713     >
3714     {

```

At the beginning of the cell, we link `\CodeAfter` to a command which begins with `\\` (whereas the standard version of `\CodeAfter` begins does not).

```

3715     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3716     \bool_gset_true:N \g_@@_after_col_zero_bool
3717     \@@_begin_of_row:
3718     \hbox_set:Nw \l_@@_cell_box
3719     \@@_math_toggle:
3720     \@@_tuning_key_small:

```

We insert `\l_@@_code_for_first_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

3721     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3722     {
3723         \bool_lazy_or:nnT
3724         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3725         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3726         {
3727             \l_@@_code_for_first_col_tl
3728             \xglobal \colorlet { nicematrix-first-col } { . }
3729         }
3730     }
3731 }

```

Be careful: despite this letter `l` the cells of the “first column” are composed in a `R` manner since they are composed in a `\hbox_overlap_left:n`.

```

3732     l
3733     <
3734     {
3735         \@@_math_toggle:
3736         \hbox_set_end:
3737         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3738         \@@_adjust_size_box:
3739         \@@_update_for_first_and_last_row:

```

We actualise the width of the “first column” because we will use this width after the construction of the array.

```

3740     \dim_gset:Nn \g_@@_width_first_col_dim
3741     { \dim_max:nn { \g_@@_width_first_col_dim } { \box_wd:N \l_@@_cell_box } }

```

The content of the cell is inserted in an overlapping position.

```

3742     \hbox_overlap_left:n
3743     {

```

```

3744     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3745     { \@@_node_cell: }
3746     { \box_use_drop:N \l_@@_cell_box }
3747     \skip_horizontal:N \l_@@_left_delim_dim
3748     \skip_horizontal:N \l_@@_left_margin_dim
3749     \skip_horizontal:N \l_@@_extra_left_margin_dim
3750   }
3751   \bool_gset_false:N \g_@@_empty_cell_bool
3752   \skip_horizontal:n { -2 \col@sep }
3753 }
3754 }

```

Here is the preamble for the “last column” (if the user uses the key `last-col`).

```

3755 \tl_const:Nn \c_@@_preamble_last_col_tl
3756 {
3757   >
3758   {
3759     \bool_set_true:N \l_@@_in_last_col_bool

```

At the beginning of the cell, we link `\CodeAfter` to a command which begins with `\` (whereas the standard version of `\CodeAfter` begins does not).

```

3760     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

3761     \bool_gset_true:N \g_@@_last_col_found_bool
3762     \int_gincr:N \c@jCol
3763     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3764     \hbox_set:Nw \l_@@_cell_box
3765     \@@_math_toggle:
3766     \@@_tuning_key_small:

```

We insert `\l_@@_code_for_last_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

3767     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3768     {
3769       \bool_lazy_or:nnT
3770       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3771       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3772       {
3773         \l_@@_code_for_last_col_tl
3774         \xglobal \colorlet { nicematrix-last-col } { . }
3775       }
3776     }
3777   }
3778   1
3779   <
3780   {
3781     \@@_math_toggle:
3782     \hbox_set_end:
3783     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3784     \@@_adjust_size_box:
3785     \@@_update_for_first_and_last_row:

```

We actualise the width of the “last column” because we will use this width after the construction of the array.

```

3786     \dim_gset:Nn \g_@@_width_last_col_dim
3787     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3788     \skip_horizontal:n { -2 \col@sep }

```

The content of the cell is inserted in an overlapping position.

```

3789     \hbox_overlap_right:n
3790     {
3791       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3792       {
3793         \skip_horizontal:N \l_@@_right_delim_dim
3794         \skip_horizontal:N \l_@@_right_margin_dim

```



```

3795         \skip_horizontal:N \l_@@_extra_right_margin_dim
3796         \@@_node_cell:
3797     }
3798 }
3799 \bool_gset_false:N \g_@@_empty_cell_bool
3800 }
3801 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}`.

```

3802 \NewDocumentEnvironment { NiceArray } { }
3803 {
3804     \bool_gset_false:N \g_@@_delims_bool
3805     \str_if_empty:NT \g_@@_name_env_str
3806     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put `.` and `.` for the delimiters but, in fact, that doesn't matter because these arguments won't be used in `{NiceArrayWithDelims}` (because the flag `\g_@@_delims_bool` is set to false).

```

3807     \NiceArrayWithDelims . .
3808 }
3809 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

3810 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3811 {
3812     \NewDocumentEnvironment { #1 NiceArray } { }
3813     {
3814         \bool_gset_true:N \g_@@_delims_bool
3815         \str_if_empty:NT \g_@@_name_env_str
3816         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3817         \@@_test_if_math_mode:
3818         \NiceArrayWithDelims #2 #3
3819     }
3820     { \endNiceArrayWithDelims }
3821 }
3822 \@@_def_env:NNN p ( )
3823 \@@_def_env:NNN b [ ]
3824 \@@_def_env:NNN B \{ \}
3825 \@@_def_env:NNN v \vert \vert
3826 \@@_def_env:NNN V \Vert \Vert

```

## 13 The environment `{NiceMatrix}` and its variants

```

3827 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3828 {
3829     \bool_set_false:N \l_@@_preamble_bool
3830     \tl_clear:N \l_tmpa_tl
3831     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3832     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3833     \tl_put_right:Nn \l_tmpa_tl
3834     {
3835         *
3836         {
3837             \int_case:nnF \l_@@_last_col_int
3838             {
3839                 { -2 } { \c@MaxMatrixCols }
3840                 { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3841     }
3842     { \int_eval:n { \l_@@_last_col_int - 1 } }
3843   }
3844   { #2 }
3845 }
3846 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3847 \exp_args:No \l_tmpb_tl \l_tpa_tl
3848 }
3849 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3850 \clist_map_inline:nn { p , b , B , v , V }
3851 {
3852   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3853   {
3854     \bool_gset_true:N \g_@@_delims_bool
3855     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3856     \int_if_zero:nT { \l_@@_last_col_int }
3857     {
3858       \bool_set_true:N \l_@@_last_col_without_value_bool
3859       \int_set:Nn \l_@@_last_col_int { -1 }
3860     }
3861     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3862     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3863   }
3864   { \use:c { end #1 NiceArray } }
3865 }

```

We define also an environment {NiceMatrix}

```

3866 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3867 {
3868   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3869   \int_if_zero:nT { \l_@@_last_col_int }
3870   {
3871     \bool_set_true:N \l_@@_last_col_without_value_bool
3872     \int_set:Nn \l_@@_last_col_int { -1 }
3873   }
3874   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3875   \bool_lazy_or:nnT
3876   { \clist_if_empty_p:N \l_@@_vlines_clist }
3877   { \l_@@_except_borders_bool }
3878   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3879   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3880 }
3881 { \endNiceArray }

```

The following command will be linked to \NotEmpty in the environments of nicematrix.

```

3882 \cs_new_protected:Npn \@@_NotEmpty:
3883 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## 14 {NiceTabular}, {NiceTabularX} and {NiceTabular\*}

```

3884 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3885 {

```

If the dimension \l\_@@\_width\_dim is equal to 0 pt, that means that it has not been set by a previous use of \NiceMatrixOptions.

```

3886   \dim_compare:nNt { \l_@@_width_dim } = { \c_zero_dim }
3887   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3888   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3889   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3890   \tl_if_empty:NF \l_@@_short_caption_tl

```

```

3891 {
3892   \tl_if_empty:NT \l_@@_caption_tl
3893   {
3894     \@@_error_or_warning:n { short-caption-without~caption }
3895     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3896   }
3897 }
3898 \tl_if_empty:NF \l_@@_label_tl
3899 {
3900   \tl_if_empty:NT \l_@@_caption_tl
3901   { \@@_error_or_warning:n { label~without~caption } }
3902 }
3903 \NewDocumentEnvironment { TabularNote } { b }
3904 {
3905   \bool_if:NTF \l_@@_in_code_after_bool
3906   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3907   {
3908     \tl_if_empty:NF \g_@@_tabularnote_tl
3909     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3910     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3911   }
3912 }
3913 { }
3914 \@@_settings_for_tabular:
3915 \NiceArray { #2 }
3916 }
3917 { \endNiceArray }
3918 \cs_new_protected:Npn \@@_settings_for_tabular:
3919 {
3920   \bool_set_true:N \l_@@_tabular_bool
3921   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3922   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3923   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3924 }

3925 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3926 {
3927   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3928   \dim_set:Nn \l_@@_width_dim { #1 }
3929   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3930   \@@_settings_for_tabular:
3931   \NiceArray { #3 }
3932 }
3933 {
3934   \endNiceArray
3935   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3936   { \@@_error:n { NiceTabularX~without~X } }
3937 }

3938 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3939 {
3940   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3941   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3942   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3943   \@@_settings_for_tabular:
3944   \NiceArray { #3 }
3945 }
3946 { \endNiceArray }

```

## 15 After the construction of the array

The following command will be used when the key `rounded-corners` is in force (this is the key `rounded-corners` for the whole environment and *not* the key `rounded-corners` of a command `\Block`).

```

3947 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3948 {
3949   \bool_lazy_all:nT
3950   {
3951     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3952     { \l_@@_hvlines_bool }
3953     { ! \g_@@_delims_bool }
3954     { ! \l_@@_except_borders_bool }
3955   }
3956   {
3957     \bool_set_true:N \l_@@_except_borders_bool
3958     \clist_if_empty:NF \l_@@_corners_clist
3959     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3960     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3961     {
3962       \@@_stroke_block:nnn
3963       {
3964         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3965         draw = \l_@@_rules_color_tl
3966       }
3967       { 1-1 }
3968       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3969     }
3970   }
3971 }

3972 \cs_new_protected:Npn \@@_after_array:
3973 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3974   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3975   \group_begin:

```

When the option `last-col` is used in the environments with explicit preambles (like `{NiceArray}`, `{pNiceArray}`, etc.) a special type of column is used at the end of the preamble in order to compose the cells in an overlapping position (with `\hbox_overlap_right:n`) but (if `last-col` has been used), we don't have the number of that last column. However, we have to know that number for the color of the potential `\Vdots` drawn in that last column. That's why we fix the correct value of `\l_@@_last_col_int` in that case.

```

3976   \bool_if:NT \g_@@_last_col_found_bool
3977   { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we also fix the real value of `\l_@@_last_col_int`.

```

3978   \bool_if:NT \l_@@_last_col_without_value_bool
3979   { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

It's also time to give to `\l_@@_last_row_int` its real value.

```

3980   \bool_if:NT \l_@@_last_row_without_value_bool
3981   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3982 \tl_gput_right:Ne \g_@@_aux_tl
3983 {
3984   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3985   {
3986     \int_use:N \l_@@_first_row_int ,
3987     \int_use:N \c@iRow ,
3988     \int_use:N \g_@@_row_total_int ,
3989     \int_use:N \l_@@_first_col_int ,
3990     \int_use:N \c@jCol ,
3991     \int_use:N \g_@@_col_total_int
3992   }
3993 }
3994 \clist_if_empty:NF \g_@@_cbic_clist { \@@_cbic: }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq`. It will be used to recreate the blocks with a name in the `\CodeBefore` and also if the command `\rowcolors` is used with the key `respect-blocks`).

```

3995 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3996 {
3997   \tl_gput_right:Ne \g_@@_aux_tl
3998   {
3999     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4000     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4001   }
4002 }
4003 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4004 {
4005   \tl_gput_right:Ne \g_@@_aux_tl
4006   {
4007     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4008     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4009     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4010     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4011   }
4012 }

```

Now, you create the diagonal nodes by using the `row` nodes and the `col` nodes.

```

4013 \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

4014 \pgfpicture
4015 \@@_create_aliases_last:
4016 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
4017 \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>12</sup>. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

4018 \bool_if:NT \l_@@_parallelize_diags_bool
4019 {
4020   \int_gzero:N \g_@@_ddots_int
4021   \int_gzero:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  and  $\Delta_y$  of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the  $\Delta_x$  and  $\Delta_y$  of the first `\Iddots` diagonal.

```

4022 \dim_gzero:N \g_@@_delta_x_one_dim
4023 \dim_gzero:N \g_@@_delta_y_one_dim
4024 \dim_gzero:N \g_@@_delta_x_two_dim

```

---

<sup>12</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.

```

4025     \dim_gzero:N \g_@@_delta_y_two_dim
4026   }
4027   \bool_set_false:N \l_@@_initial_open_bool
4028   \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

4029   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

4030   \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

4031   \clist_if_empty:NF \l_@@_corners_clist
4032   {
4033     \bool_if:NTF \l_@@_no_cell_nodes_bool
4034     { \@@_error:n { corners~with~no~cell~nodes } }
4035     { \@@_compute_corners: }
4036   }

```

By design, we have computed the corners before the adjonction of `\g_@@_future_pos_of_blocks_seq` is used by `\EmptyRow` and `\EmptyColumn` in the `\CodeBefore`.

```

4037   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4038   \g_@@_pos_of_blocks_seq
4039   \g_@@_future_pos_of_blocks_seq
4040   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

4041   \@@_adjust_pos_of_blocks_seq:
4042   \@@_deal_with_rounded_corners:
4043   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
4044   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

Now, the pre-code-after and then, the `\CodeAfter`.

```

4045   \IfPackageLoadedT { tikz }
4046   {
4047     \tikzset
4048     {
4049       every-picture / .style =
4050       {
4051         overlay ,
4052         remember~picture ,
4053         name~prefix = \@@_env: -
4054       }
4055     }
4056   }
4057   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4058   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4059   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4060   \cs_set_eq:NN \OverBrace \@@_OverBrace
4061   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4062   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4063   \cs_set_eq:NN \line \@@_line

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```
4064 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
4065 \tl_gclear:N \g_@@_pre_code_after_tl
```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\CodeAfter` to be *no-op* now.

```
4066 \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

We clear the list of the names of the potential `\SubMatrix` that will appear in the `\CodeAfter` (unfortunately, that list has to be global).

```
4067 \seq_gclear:N \g_@@_submatrix_names_seq
```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `>` and `<` are activated and TikZ is not able to solve the problem (even with the TikZ library `babel`).

```
4068 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
4069 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```
4070 \bool_set_true:N \l_@@_in_code_after_bool
4071 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4072 \scan_stop:
4073 \tl_gclear:N \g_nicematrix_code_after_tl
4074 \group_end:
```

`\g_@@_pre_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor`. These instructions will be written on the aux file to be added to the `code-before` in the next run.

```
4075 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4076 \tl_if_empty:NF \g_@@_pre_code_before_tl
4077 {
4078   \tl_gput_right:Ne \g_@@_aux_tl
4079   {
4080     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4081     { \exp_not:o \g_@@_pre_code_before_tl }
4082   }
4083   \tl_gclear:N \g_@@_pre_code_before_tl
4084 }
4085 \tl_if_empty:NF \g_nicematrix_code_before_tl
4086 {
4087   \tl_gput_right:Ne \g_@@_aux_tl
4088   {
4089     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4090     { \exp_not:o \g_nicematrix_code_before_tl }
4091   }
4092   \tl_gclear:N \g_nicematrix_code_before_tl
4093 }

4094 \str_gclear:N \g_@@_name_env_str
4095 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>13</sup>. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```
4096 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4097 }
```

---

<sup>13</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

4098 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4099 {
4100   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4101   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

4102   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4103     { 0.6 \l_@@_xdots_shorten_start_dim }
4104   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4105     { 0.6 \l_@@_xdots_shorten_end_dim }
4106 }

```

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```

4107 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4108 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4109 \cs_new_protected:Npn \@@_create_alias_nodes:
4110 {
4111   \int_step_inline:nn { \c@iRow }
4112   {
4113     \pgfnodealias
4114       { \l_@@_name_str - ##1 - last }
4115       { \@@_env: - ##1 - \int_use:N \c@jCol }
4116   }
4117   \int_step_inline:nn { \c@jCol }
4118   {
4119     \pgfnodealias
4120       { \l_@@_name_str - last - ##1 }
4121       { \@@_env: - \int_use:N \c@iRow - ##1 }
4122   }
4123   \pgfnodealias
4124     { \l_@@_name_str - last - last }
4125     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4126 }

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format  $i$ - $j$ . However, the user is allowed to omit  $i$  or  $j$  (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It’s possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

4127 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4128 {
4129   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4130     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4131 }

```

The following command must *not* be protected.

```

4132 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4133 {
4134   { #1 }
4135   { #2 }
4136   {
4137     \int_compare:nNnTF { #3 } > { 98 }
4138       { \int_use:N \c@iRow }
4139       { #3 }
4140   }

```



```

4141 {
4142   \int_compare:nNnTF { #4 } > { 98 }
4143     { \int_use:N \c@jCol }
4144     { #4 }
4145 }
4146 { #5 }
4147 }

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we have to define the adequate version of `\@@_draw_dotted_lines`: whether TikZ is loaded or not (in that case, only PGF is loaded).

```

4148 \hook_gput_code:nnn { begindocument } { . }
4149 {
4150   \cs_new_protected:Npe \@@_draw_dotted_lines:
4151     {
4152       \c_@@_pgfortikzpicture_tl
4153       \@@_draw_dotted_lines_i:
4154       \c_@@_endpgfortikzpicture_tl
4155     }
4156 }

```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```

4157 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4158 {
4159   \pgfrememberpicturepositiononpagetrue
4160   \pgf@relevantforpicturesizefalse
4161   \g_@@_HVdotsfor_lines_tl
4162   \g_@@_Vdots_lines_tl
4163   \g_@@_Ddots_lines_tl
4164   \g_@@_Iddots_lines_tl
4165   \g_@@_Cdots_lines_tl
4166   \g_@@_Ldots_lines_tl
4167 }

4168 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4169 {
4170   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4171   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4172 }

```

We define a new PGF shape for the diag nodes because we want to provide an anchor called `.5` for those nodes.

```

4173 \pgfdeclareshape { @@_diag_node }
4174 {
4175   \savedanchor { \five }
4176   {
4177     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4178     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4179   }
4180   \anchor { 5 } { \five }
4181   \anchor { center } { \pgfpointorigin }
4182   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4183   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4184   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4185   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4186   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4187   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4188   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4189   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4190   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4191   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4192 }

```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```

4193 \cs_new_protected:Npn \@@_create_diag_nodes:
4194 {
4195   \pgfpicture
4196   \pgfrememberpicturepositiononpagetrue
4197   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4198   {
4199     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4200     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4201     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4202     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4203     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4204     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4205     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4206     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4207     \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```

4208     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4209     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4210     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4211     \str_if_empty:NF \l_@@_name_str
4212     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4213   }

```

Now, the last node. Of course, that is only a coordinate because there is not `.5` anchor for that node.

```

4214   \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4215   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4216   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4217   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4218   \pgfcoordinate
4219   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4220   \pgfnodealias
4221   { \@@_env: - last }
4222   { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4223   \str_if_empty:NF \l_@@_name_str
4224   {
4225     \pgfnodealias
4226     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4227     { \@@_env: - \int_use:N \l_tmpa_int }
4228     \pgfnodealias
4229     { \l_@@_name_str - last }
4230     { \@@_env: - last }
4231   }
4232   \endpgfpicture
4233 }

```

## 16 We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

$$\begin{pmatrix} a+b+c & a+b & a \\ a & \cdots & \cdots \\ a & a+b & a+b+c \end{pmatrix}$$

The command `\l_@@_find_extremities:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the  $x$ -value of the orientation vector of the line;
- the fourth argument is the  $y$ -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \l_@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
  {
    \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_final_open_bool }
    }
    {
      \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
      {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
{
  \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
```

```

{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_final_i_int -
    \int_use:N \l_@@_final_j_int
  }
  {
    \int_sub:Nn \l_@@_final_i_int { #3 }
    \int_sub:Nn \l_@@_final_j_int { #4 }
    \bool_set_true:N \l_@@_final_open_bool
    \bool_set_true:N \l_@@_stop_loop_bool
  }
  {
    \cs_if_exist:cTF
    {
      pgf @ sh @ ns @ \@@_env:
      - \int_use:N \l_@@_final_i_int
      - \int_use:N \l_@@_final_j_int
    }
    { \bool_set_true:N \l_@@_stop_loop_bool }
    {
      \cs_set_nopar:cpn
      {
        @@ _ dotted _
        \int_use:N \l_@@_final_i_int -
        \int_use:N \l_@@_final_j_int
      }
      { }
    }
  }
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
  \int_sub:Nn \l_@@_initial_i_int { #3 }
  \int_sub:Nn \l_@@_initial_j_int { #4 }
  \bool_set_false:N \l_@@_initial_open_bool
  \int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
  {
    \int_compare:nNnTF { #3 } = { 1 }
    { \bool_set_true:N \l_@@_initial_open_bool }
    {
      \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
      { \bool_set_true:N \l_@@_initial_open_bool }
    }
  }
}
{
  \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
  {
    \int_compare:nNnT { #4 } = { 1 }
    { \bool_set_true:N \l_@@_initial_open_bool }
  }
  {
    \int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_initial_open_bool }
    }
  }
}
}

```

```

\bool_if:NTF \l_@@_initial_open_bool
{
  \int_add:Nn \l_@@_initial_i_int { #3 }
  \int_add:Nn \l_@@_initial_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_initial_i_int -
    \int_use:N \l_@@_initial_j_int
  }
  {
    \int_add:Nn \l_@@_initial_i_int { #3 }
    \int_add:Nn \l_@@_initial_j_int { #4 }
    \bool_set_true:N \l_@@_initial_open_bool
    \bool_set_true:N \l_@@_stop_loop_bool
  }
  {
    \cs_if_exist:cTF
    {
      pgf @ sh @ ns @ \@@_env:
      - \int_use:N \l_@@_initial_i_int
      - \int_use:N \l_@@_initial_j_int
    }
    { \bool_set_true:N \l_@@_stop_loop_bool }
    {
      \cs_set_nopar:cpn
      {
        @@ _ dotted _
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int
      }
      { }
    }
  }
}
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
  { \int_use:N \l_@@_initial_i_int }
  { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { \int_use:N \l_@@_final_i_int }
  { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { }
}
}

```

The following version is slightly more efficient.

```

4234 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4235 {

```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```

4236   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4237   \l_@@_initial_i_int = #1
4238   \l_@@_initial_j_int = #2
4239   \l_@@_final_i_int = #1
4240   \l_@@_final_j_int = #2

```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

4241     \let \l_@@_stop_loop_bool \c_false_bool
4242     \bool_do_until:Nn \l_@@_stop_loop_bool
4243     {

```

We test if we are still in the matrix.

```

4244         \advance \l_@@_final_i_int by #3
4245         \advance \l_@@_final_j_int by #4
4246         \let \l_@@_final_open_bool \c_false_bool
4247         \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4248             \if_int_compare:w #3 = \c_one_int
4249                 \let \l_@@_final_open_bool \c_true_bool
4250             \else:
4251                 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4252                     \let \l_@@_final_open_bool \c_true_bool
4253             \fi:
4254         \fi:
4255     \else:
4256         \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4257             \if_int_compare:w #4 = -1
4258                 \let \l_@@_final_open_bool \c_true_bool
4259             \fi:
4260         \else:
4261             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4262                 \if_int_compare:w #4 = \c_one_int
4263                     \let \l_@@_final_open_bool \c_true_bool
4264             \fi:
4265         \fi:
4266     \fi:
4267 \fi:
4268 \bool_if:NTF \l_@@_final_open_bool

```

If we are outside the matrix, we have found the extremity of the dotted line and it’s an *open* extremity.

```

4269     {

```

We do a step backwards.

```

4270         \advance \l_@@_final_i_int by - #3
4271         \advance \l_@@_final_j_int by - #4
4272         \let \l_@@_stop_loop_bool \c_true_bool
4273     }

```

If we are in the matrix, we test whether the cell is empty. If it’s not the case, we stop the loop because we have found the correct values for `\l_@@_final_i_int` and `\l_@@_final_j_int`.

```

4274     {
4275         \cs_if_exist:cTF
4276         {
4277             @@ _ dotted _
4278             \int_use:N \l_@@_final_i_int -
4279             \int_use:N \l_@@_final_j_int
4280         }
4281         {
4282             \advance \l_@@_final_i_int by - #3
4283             \advance \l_@@_final_j_int by - #4
4284             \let \l_@@_final_open_bool \c_true_bool
4285             \let \l_@@_stop_loop_bool \c_true_bool
4286         }
4287     {
4288         \cs_if_exist:cTF
4289         {
4290             pgf @ sh @ ns @ \@@_env:
4291             - \int_use:N \l_@@_final_i_int
4292             - \int_use:N \l_@@_final_j_int
4293         }

```

```

4294         { \let \l_@@_stop_loop_bool \c_true_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4295     {
4296         \cs_set_nopar:cpn
4297         {
4298             @@ _ dotted _
4299             \int_use:N \l_@@_final_i_int -
4300             \int_use:N \l_@@_final_j_int
4301         }
4302     { }
4303 }
4304 }
4305 }
4306 }

```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```

4307     \let \l_@@_stop_loop_bool \c_false_bool

```

The following line of code is only for efficiency in the following loop.

```

4308     \l_tmpa_int = \l_@@_col_min_int
4309     \advance \l_tmpa_int by -1
4310     \bool_do_until:Nn \l_@@_stop_loop_bool
4311     {
4312         \advance \l_@@_initial_i_int by - #3
4313         \advance \l_@@_initial_j_int by - #4
4314         \let \l_@@_initial_open_bool \c_false_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4315         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4316         \if_int_compare:w #3 = \c_one_int
4317             \let \l_@@_initial_open_bool \c_true_bool
4318         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4319             \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4320                 \let \l_@@_initial_open_bool \c_true_bool
4321             \fi:
4322         \fi:
4323     \else:
4324         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4325         \if_int_compare:w #4 = \c_one_int
4326             \let \l_@@_initial_open_bool \c_true_bool
4327         \fi:
4328     \else:
4329         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4330         \if_int_compare:w #4 = -1
4331             \let \l_@@_initial_open_bool \c_true_bool
4332         \fi:
4333     \fi:
4334 \fi:
4335 \fi:
4336 \bool_if:NTF \l_@@_initial_open_bool
4337 {
4338     \advance \l_@@_initial_i_int by #3
4339     \advance \l_@@_initial_j_int by #4

```

```

4340     \let \l_@@_stop_loop_bool \c_true_bool
4341   }
4342   {
4343     \cs_if_exist:cTF
4344     {
4345       @@ _ dotted _
4346       \int_use:N \l_@@_initial_i_int -
4347       \int_use:N \l_@@_initial_j_int
4348     }
4349     {
4350       \advance \l_@@_initial_i_int by #3
4351       \advance \l_@@_initial_j_int by #4
4352       \let \l_@@_initial_open_bool \c_true_bool
4353       \let \l_@@_stop_loop_bool \c_true_bool
4354     }
4355     {
4356       \cs_if_exist:cTF
4357       {
4358         pgf @ sh @ ns @ \@@_env:
4359         - \int_use:N \l_@@_initial_i_int
4360         - \int_use:N \l_@@_initial_j_int
4361       }
4362       { \let \l_@@_stop_loop_bool \c_true_bool }
4363       {
4364         \cs_set_nopar:cpn
4365         {
4366           @@ _ dotted _
4367           \int_use:N \l_@@_initial_i_int -
4368           \int_use:N \l_@@_initial_j_int
4369         }
4370         { }
4371       }
4372     }
4373   }
4374 }

```

We remind the rectangle described by all the dotted lines in order to respect the corresponding virtual “block” when drawing the horizontal and vertical rules.

```

4375   \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4376   {
4377     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4378     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4379     { \int_use:N \l_@@_final_i_int }
4380     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4381     { }
4382   }
4383 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we known whether the extremities are closed or open) but before the analysis of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4384 \cs_new_protected:Npn \@@_open_shorten:
4385 {
4386   \bool_if:NT \l_@@_initial_open_bool
4387   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4388   \bool_if:NT \l_@@_final_open_bool
4389   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4390 }

```



The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it's only the whole array (excepted exterior rows and columns).

```

4391 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4392 {
4393   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4394   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4395   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4396   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the `code-before`. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

4397   \seq_if_empty:NF \g_@@_submatrix_seq
4398   {
4399     \seq_map_inline:Nn \g_@@_submatrix_seq
4400     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4401   }
4402 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4403 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4404 {
4405   \if_int_compare:w #3 > #1
4406   \else:
4407     \if_int_compare:w #1 > #5
4408     \else:
4409       \if_int_compare:w #4 > #2
4410       \else:
4411         \if_int_compare:w #2 > #6
4412         \else:
4413           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4414           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4415           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4416           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4417         \fi:
4418       \fi:
4419     \fi:
4420   \fi:
4421 }

```

```

4422 \cs_new_protected:Npn \@@_set_initial_coords:
4423 {
4424   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4425   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4426 }
4427 \cs_new_protected:Npn \@@_set_final_coords:
4428 {
4429   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4430   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4431 }
4432 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4433 {
4434   \pgfpointanchor
4435   {
4436     \@@_env:
4437     - \int_use:N \l_@@_initial_i_int
4438     - \int_use:N \l_@@_initial_j_int
4439   }
4440   { #1 }
4441   \@@_set_initial_coords:
4442 }
4443 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4444 {
4445   \pgfpointanchor
4446   {
4447     \@@_env:
4448     - \int_use:N \l_@@_final_i_int
4449     - \int_use:N \l_@@_final_j_int
4450   }
4451   { #1 }
4452   \@@_set_final_coords:
4453 }
4454 \cs_new_protected:Npn \@@_open_x_initial_dim:
4455 {
4456   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4457   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4458   {
4459     \cs_if_exist:cT
4460     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4461     {
4462       \pgfpointanchor
4463       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4464       { west }
4465       \dim_set:Nn \l_@@_x_initial_dim
4466       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4467     }
4468   }

```

If, in fact, all the cells of the column are empty (no PGF/TikZ nodes in those cells).

```

4469   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4470   {
4471     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4472     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4473     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4474   }
4475 }
4476 \cs_new_protected:Npn \@@_open_x_final_dim:
4477 {
4478   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4479   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4480   {
4481     \cs_if_exist:cT
4482     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4483     {
4484         \pgfpointanchor
4485         { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4486         { east }
4487         \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4488         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4489     }
4490 }

```

If, in fact, all the cells of the columns are empty (no PGF/TikZ nodes in those cells).

```

4491     \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4492     {
4493         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4494         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4495         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4496     }
4497 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4498 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4499 {
4500     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4501     \cs_if_free:cT { @ _ dotted _ #1 - #2 }
4502     {
4503         \l_@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4504         \bool_if:NT \g_@@_aux_found_bool
4505         {
4506             \group_begin:
4507             \l_@@_open_shorten:
4508             \int_if_zero:nTF { #1 }
4509             { \color { nicematrix-first-row } }
4510         }

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4511             \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4512             { \color { nicematrix-last-row } }
4513         }
4514         \keys_set:nn { nicematrix / xdots } { #3 }
4515         \l_@@_color:o \l_@@_xdots_color_tl
4516         \l_@@_actually_draw_Ldots:
4517     \group_end:
4518 }
4519 }
4520 }

```

The command `\l_@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Hdotsfor`.

```

4521 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4522 {
4523   \bool_if:NTF \l_@@_initial_open_bool
4524   {
4525     \@@_open_x_initial_dim:
4526     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4527     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4528   }
4529   { \@@_set_initial_coords_from_anchor:n { base-east } }
4530 \bool_if:NTF \l_@@_final_open_bool
4531 {
4532   \@@_open_x_final_dim:
4533   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4534   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4535 }
4536 { \@@_set_final_coords_from_anchor:n { base-west } }

```

Now the case of a `\Hdotsfor` (or when there is only a `\Ldots`) in the “last row” (that case will probably arise when the final user draws an arrow to indicate the number of columns of the matrix). In the “first row”, we don’t need any adjustment.

```

4537 \bool_lazy_all:nTF
4538 {
4539   \l_@@_initial_open_bool
4540   \l_@@_final_open_bool
4541   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4542 }
4543 {
4544   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4545   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4546 }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4547 {
4548   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4549   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4550 }
4551 \@@_draw_line:
4552 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4553 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4554 {
4555   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4556   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4557   {
4558     \@@_find_extremities:nnnn { #1 } { #2 } { 0 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4559 \bool_if:NT \g_@@_aux_found_bool
4560 {
4561   \group_begin:
4562   \@@_open_shorten:
4563   \int_if_zero:nTF { #1 }
4564   { \color { nicematrix-first-row } }
4565   {

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4566 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }

```

```

4567         { \color { nicematrix-last-row } }
4568     }
4569     \keys_set:nn { nicematrix / xdots } { #3 }
4570     \@@_color:o \l_@@_xdots_color_tl
4571     \@@_actually_draw_Cdots:
4572     \group_end:
4573 }
4574 }
4575 }

```

The command `\@@_actually_draw_Cdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

```

4576 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4577 {
4578     \bool_if:NTF \l_@@_initial_open_bool
4579     { \@@_open_x_initial_dim: }
4580     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4581     \bool_if:NTF \l_@@_final_open_bool
4582     { \@@_open_x_final_dim: }
4583     { \@@_set_final_coords_from_anchor:n { mid-west } }
4584     \bool_lazy_and:nnTF
4585     { \l_@@_initial_open_bool }
4586     { \l_@@_final_open_bool }
4587     {
4588         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4589         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4590         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4591         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4592         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4593     }
4594     {
4595         \bool_if:NT \l_@@_initial_open_bool
4596         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4597         \bool_if:NT \l_@@_final_open_bool
4598         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4599     }
4600     \@@_draw_line:
4601 }
4602 \cs_new_protected:Npn \@@_open_y_initial_dim:
4603 {
4604     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4605     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4606     {
4607         \cs_if_exist:cT
4608         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4609         {
4610             \pgfpointanchor
4611             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4612             { north }
4613             \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4614             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4615         }
4616     }

```

```

4617 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4618 {
4619   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4620   \dim_set:Nn \l_@@_y_initial_dim
4621   {
4622     \fp_to_dim:n
4623     {
4624       \pgf@y
4625       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4626     }
4627   }
4628 }
4629 }
4630 \cs_new_protected:Npn \@@_open_y_final_dim:
4631 {
4632   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4633   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4634   {
4635     \cs_if_exist:cT
4636     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4637     {
4638       \pgfpointanchor
4639       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4640       { south }
4641       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4642       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4643     }
4644   }
4645   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4646   {
4647     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4648     \dim_set:Nn \l_@@_y_final_dim
4649     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4650   }
4651 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4652 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4653 {
4654   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4655   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4656   {
4657     \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { 0 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4658   \bool_if:NT \g_@@_aux_found_bool
4659   {
4660     \group_begin:
4661     \@@_open_shorten:
4662     \int_if_zero:nTF { #2 }
4663     { \color { nicematrix-first-col } }
4664     {
4665       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4666       { \color { nicematrix-last-col } }
4667     }
4668     \keys_set:nn { nicematrix / xdots } { #3 }
4669     \@@_color:o \l_@@_xdots_color_tl
4670     \bool_if:NTF \l_@@_Vbrace_bool
4671     { \@@_actually_draw_Vbrace: }
4672     { \@@_actually_draw_Vdots: }
4673     \group_end:
4674   }

```

```

4675     }
4676 }

```

The following function is used by regular calls of `\Vdots` or `\Vdotsfor` but not by `\Vbrace`. The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4677 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4678 {
4679   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4680   { \@@_actually_draw_Vdots_i: }
4681   { \@@_actually_draw_Vdots_ii: }
4682   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4683   \@@_draw_line:
4684 }

```

First, the case of a dotted line open on both sides.

```

4685 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4686 {
4687   \@@_open_y_initial_dim:
4688   \@@_open_y_final_dim:
4689   \int_if_zero:nTF { \l_@@_initial_j_int }

```

We have a dotted line open on both sides in the “first column”.

```

4690   {
4691     \@@_qpoint:n { col - 1 }
4692     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4693     \dim_sub:Nn \l_@@_x_initial_dim
4694     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4695   }
4696   {
4697     \bool_lazy_and:nnTF
4698     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4699     {
4700       \int_compare_p:nNn
4701       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4702     }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4703   {
4704     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4705     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4706     \dim_add:Nn \l_@@_x_initial_dim
4707     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4708   }

```

We have a dotted line open on both sides which is *not* in an exterior column.

```

4709   {
4710     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4711     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4712     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4713     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4714   }
4715 }
4716 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the  $x$ -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```

4717 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4718 {
4719   \bool_set_false:N \l_tmpa_bool
4720   \bool_if:NF \l_@@_initial_open_bool
4721   {
4722     \bool_if:NF \l_@@_final_open_bool
4723     {
4724       \@@_set_initial_coords_from_anchor:n { south~west }
4725       \@@_set_final_coords_from_anchor:n { north~west }
4726       \bool_set:Nn \l_tmpa_bool
4727       {
4728         \dim_compare_p:nNn
4729         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4730       }
4731     }
4732   }

```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```

4733   \bool_if:NTF \l_@@_initial_open_bool
4734   {
4735     \@@_open_y_initial_dim:
4736     \@@_set_final_coords_from_anchor:n { north }
4737     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4738   }
4739   {
4740     \@@_set_initial_coords_from_anchor:n { south }
4741     \bool_if:NTF \l_@@_final_open_bool
4742     { \@@_open_y_final_dim: }

```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` or may be considered as if.

```

4743   {
4744     \@@_set_final_coords_from_anchor:n { north }
4745     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4746     {
4747       \dim_set:Nn \l_@@_x_initial_dim
4748       {
4749         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4750         \l_@@_x_initial_dim \l_@@_x_final_dim
4751       }
4752     }
4753   }
4754 }
4755 }

```

The following function is used by `\Vbrace` but not by regular uses of `\Vdots` or `\Vdotsfor`.

The command `\@@_actually_draw_Vbrace:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.



```

4756 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4757 {
4758   \bool_if:NTF \l_@@_initial_open_bool
4759     { \@@_open_y_initial_dim: }
4760     { \@@_set_initial_coords_from_anchor:n { south } }
4761   \bool_if:NTF \l_@@_final_open_bool
4762     { \@@_open_y_final_dim: }
4763     { \@@_set_final_coords_from_anchor:n { north } }

```

Now, we have the correct values for the  $y$ -values of both extremities of the brace. We have to compute the  $x$ -value (there is only one  $x$ -value since, of course, the brace is vertical).

If we are in the first (exterior) column, the brace must be drawn right flush.

```

4764   \int_if_zero:nTF { \l_@@_initial_j_int }
4765   {
4766     \@@_qpoint:n { col - 1 }
4767     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4768     \dim_sub:Nn \l_@@_x_initial_dim
4769       { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4770   }

```

Elsewhere, the brace must be drawn left flush.

```

4771   {
4772     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4773     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4774     \dim_add:Nn \l_@@_x_initial_dim
4775       { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4776   }

```

We draw a vertical rule and that's why, of course, both  $x$ -values are equal.

```

4777   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4778   \@@_draw_line:
4779 }

```

```

4780 \cs_new:Npn \@@_colsep:
4781 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4782 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4783 {
4784   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4785   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4786   {
4787     \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4788   \bool_if:NT \g_@@_aux_found_bool
4789   {
4790     \group_begin:
4791       \@@_open_shorten:
4792       \keys_set:nn { nicematrix / xdots } { #3 }
4793       \@@_color:o \l_@@_xdots_color_tl
4794       \@@_actually_draw_Ddots:
4795     \group_end:
4796   }
4797 }
4798 }

```

The command `\l_@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

```

4799 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4800 {
4801   \bool_if:NTF \l_@@_initial_open_bool
4802   {
4803     \@@_open_y_initial_dim:
4804     \@@_open_x_initial_dim:
4805   }
4806   { \@@_set_initial_coords_from_anchor:n { south~east } }
4807   \bool_if:NTF \l_@@_final_open_bool
4808   {
4809     \@@_open_x_final_dim:
4810     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4811   }
4812   { \@@_set_final_coords_from_anchor:n { north~west } }

```

We have retrieved the coordinates in the usual way (they are stored in `\l_@@_x_initial_dim`, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

```

4813   \bool_if:NT \l_@@_parallelize_diags_bool
4814   {
4815     \int_gincr:N \g_@@_ddots_int

```

We test if the diagonal line is the first one (the counter `\g_@@_ddots_int` is created for this usage).

```

4816     \int_compare:nNnTF { \g_@@_ddots_int } = { \c_one_int }

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4817     {
4818       \dim_gset:Nn \g_@@_delta_x_one_dim
4819       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4820       \dim_gset:Nn \g_@@_delta_y_one_dim
4821       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4822     }

```

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate `\l_@@_x_initial_dim`.

```

4823     {
4824       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4825       {
4826         \dim_set:Nn \l_@@_y_final_dim
4827         {
4828           \l_@@_y_initial_dim +
4829           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4830           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4831         }
4832       }
4833     }
4834   }
4835   \@@_draw_line:
4836 }

```

We draw the `\Iddots` diagonals in the same way.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4837 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4838 {
4839   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4840   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4841   {
4842     \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4843     \bool_if:NT \g_@@_aux_found_bool
4844     {
4845       \group_begin:
4846       \@@_open_shorten:
4847       \keys_set:nn { nicematrix / xdots } { #3 }
4848       \@@_color:o \l_@@_xdots_color_tl
4849       \@@_actually_draw_Iddots:
4850       \group_end:
4851     }
4852   }
4853 }

```

The command `\@@_actually_draw_Iddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4854 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4855 {
4856   \bool_if:NTF \l_@@_initial_open_bool
4857   {
4858     \@@_open_y_initial_dim:
4859     \@@_open_x_initial_dim:
4860   }
4861   { \@@_set_initial_coords_from_anchor:n { south-west } }
4862   \bool_if:NTF \l_@@_final_open_bool
4863   {
4864     \@@_open_y_final_dim:
4865     \@@_open_x_final_dim:
4866   }
4867   { \@@_set_final_coords_from_anchor:n { north-east } }
4868   \bool_if:NT \l_@@_parallelize_diags_bool
4869   {
4870     \int_gincr:N \g_@@_iddots_int
4871     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4872     {
4873       \dim_gset:Nn \g_@@_delta_x_two_dim
4874       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4875       \dim_gset:Nn \g_@@_delta_y_two_dim
4876       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4877     }
4878     {
4879       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4880       {

```

```

4881         \dim_set:Nn \l_@@_y_final_dim
4882         {
4883             \l_@@_y_initial_dim +
4884             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4885             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4886         }
4887     }
4888 }
4889 }
4890 \@@_draw_line:
4891 }

```

## 17 The actual instructions for drawing the dotted lines with TikZ

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4892 \cs_new_protected:Npn \@@_draw_line:
4893 {
4894     \pgfrememberpicturepositiononpagetrue
4895     \pgf@relevantforpicturesizefalse
4896     \bool_lazy_or:nnTF
4897     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4898     { \l_@@_dotted_bool }
4899     { \@@_draw_standard_dotted_line: }
4900     { \@@_draw_unstandard_dotted_line: }
4901 }

```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the TikZ instruction.

```

4902 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4903 {
4904     \begin { scope }
4905     \@@_draw_unstandard_dotted_line:o
4906     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4907 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4908 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4909 {
4910     \@@_draw_unstandard_dotted_line:nooo
4911     { #1 }
4912     \l_@@_xdots_up_tl
4913     \l_@@_xdots_down_tl
4914     \l_@@_xdots_middle_tl
4915 }
4916 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following TikZ styles are for the three labels (set by the symbols  $_$ ,  $\wedge$  and  $=$ ) of a continuous line with a non-standard style.

```

4917 \hook_gput_code:nnn { begindocument } { . }
4918 {
4919   \IfPackageLoadedT { tikz }
4920   {
4921     \tikzset
4922     {
4923       @@_node_above / .style = { sloped , above } ,
4924       @@_node_below / .style = { sloped , below } ,
4925       @@_node_middle / .style =
4926       {
4927         sloped ,
4928         inner-sep = \c_@@_innersep_middle_dim
4929       }
4930     }
4931   }
4932 }

4933 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4934 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4935 \dim_zero_new:N \l_@@_l_dim
4936 \dim_set:Nn \l_@@_l_dim
4937 {
4938   \fp_to_dim:n
4939   {
4940     sqrt
4941     (
4942       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4943       +
4944       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4945     )
4946   }
4947 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the `aux` file to say that one more compilation should be done.

```

4948 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4949 {
4950   \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4951   \@@_draw_unstandard_dotted_line_i:
4952 }

```

If the key `xdots/horizontal-labels` has been used.

```

4953 \bool_if:NT \l_@@_xdots_h_labels_bool
4954 {
4955   \tikzset
4956   {
4957     @@_node_above / .style = { auto = left } ,
4958     @@_node_below / .style = { auto = right } ,
4959     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4960   }
4961 }
4962 \tl_if_empty:nF { #4 }
4963 { \tikzset { @@_node_middle / .append-style = { fill = white } } }

```

```

4964 \dim_zero:N \l_tmpa_dim
4965 \dim_zero:N \l_tmpb_dim
4966 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4967 {

```

We test whether the brace is vertical or horizontal.

```

4968 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4969 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4970 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4971 }
4972 {
4973 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4974 {
4975 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4976 { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4977 { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4978 }
4979 }
4980 \use:e
4981 {
4982 \exp_not:N \begin { scope }
4983 [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
4984 }
4985 \draw
4986 [ #1 ]
4987 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4988 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4989 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4990 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4991 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4992 \end { scope }
4993 \end { scope }
4994 }
4995 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4996 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4997 {
4998 \dim_set:Nn \l_tmpa_dim
4999 {
5000 \l_@@_x_initial_dim
5001 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5002 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5003 }
5004 \dim_set:Nn \l_tmpb_dim
5005 {
5006 \l_@@_y_initial_dim
5007 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5008 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5009 }
5010 \dim_set:Nn \l_@@_tmpc_dim
5011 {
5012 \l_@@_x_final_dim
5013 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5014 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5015 }
5016 \dim_set:Nn \l_@@_tmpd_dim
5017 {
5018 \l_@@_y_final_dim
5019 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5020 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5021 }
5022 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5023 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5024 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim

```

```

5025 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5026 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

5027 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5028 {
5029 \group_begin:

```

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

5030 \dim_zero_new:N \l_@@_l_dim
5031 \dim_set:Nn \l_@@_l_dim
5032 {
5033 \fp_to_dim:n
5034 {
5035 sqrt
5036 (
5037 (\l_@@_x_final_dim - \l_@@_x_initial_dim) ^ 2
5038 +
5039 (\l_@@_y_final_dim - \l_@@_y_initial_dim) ^ 2
5040 )
5041 }
5042 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

```

5043 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
5044 {
5045 \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
5046 { \@@_draw_standard_dotted_line_i: }
5047 }
5048 \group_end:
5049 \bool_lazy_all:nF
5050 {
5051 { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5052 { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5053 { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5054 }
5055 { \@@_labels_standard_dotted_line: }
5056 }
5057 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5058 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5059 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

5060 \int_set:Nn \l_tmpa_int
5061 {
5062 \dim_ratio:nn
5063 {
5064 \l_@@_l_dim
5065 - \l_@@_xdots_shorten_start_dim
5066 - \l_@@_xdots_shorten_end_dim
5067 }
5068 { \l_@@_xdots_inter_dim }
5069 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

5070 \dim_set:Nn \l_tmpa_dim

```

```

5071 {
5072   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5073   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5074 }
5075 \dim_set:Nn \l_tmpb_dim
5076 {
5077   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5078   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5079 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5080 \dim_gadd:Nn \l_@@_x_initial_dim
5081 {
5082   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5083   \dim_ratio:nn
5084   {
5085     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5086     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5087   }
5088   { 2 \l_@@_l_dim }
5089 }
5090 \dim_gadd:Nn \l_@@_y_initial_dim
5091 {
5092   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5093   \dim_ratio:nn
5094   {
5095     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5096     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5097   }
5098   { 2 \l_@@_l_dim }
5099 }
5100 \pgf@relevantforpicturesizefalse
5101 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
5102 {
5103   \pgfpathcircle
5104   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5105   { \l_@@_xdots_radius_dim }
5106   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5107   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5108 }
5109 \pgfusepathqfill
5110 }

```

```

5111 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5112 {
5113   \pgfscope
5114   \pgftransformshift
5115   {
5116     \pgfpointlineattime { 0.5 }
5117     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5118     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5119   }
5120   \fp_set:Nn \l_tmpa_fp
5121   {
5122     atand
5123     (
5124       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5125       \l_@@_x_final_dim - \l_@@_x_initial_dim
5126     )
5127   }
5128   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5129   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }

```



```

5130 \tl_if_empty:NF \l_@@_xdots_middle_tl
5131 {
5132   \begin { pgfscope }
5133   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5134   \pgfnode
5135     { rectangle }
5136     { center }
5137     {
5138       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5139       {
5140         $ % $
5141         \scriptstyle \l_@@_xdots_middle_tl
5142         $ % $
5143       }
5144     }
5145     { }
5146     {
5147       \pgfsetfillcolor { white }
5148       \pgfusepath { fill }
5149     }
5150   \end { pgfscope }
5151 }
5152 \tl_if_empty:NF \l_@@_xdots_up_tl
5153 {
5154   \pgfnode
5155     { rectangle }
5156     { south }
5157     {
5158       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5159       {
5160         $ % $
5161         \scriptstyle \l_@@_xdots_up_tl
5162         $ % $
5163       }
5164     }
5165     { }
5166     { \pgfusepath { } }
5167 }
5168 \tl_if_empty:NF \l_@@_xdots_down_tl
5169 {
5170   \pgfnode
5171     { rectangle }
5172     { north }
5173     {
5174       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5175       {
5176         $ % $
5177         \scriptstyle \l_@@_xdots_down_tl
5178         $ % $
5179       }
5180     }
5181     { }
5182     { \pgfusepath { } }
5183 }
5184 \endpgfscope
5185 }

```

## 18 User commands available in the new environments

The commands `\@@_Ldots:`, `\@@_Cdots:`, `\@@_Vdots:`, `\@@_Ddots:` and `\@@_Iddots:` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```
5186 \hook_gput_code:nnn { begindocument } { . }
5187 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
5188 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5189 \cs_new_protected:Npn \@@_Ldots:
5190 { \@@_collect_options:n { \@@_Ldots_i } }
5191 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5192 {
5193   \int_if_zero:nTF { \c@jCol }
5194   { \@@_error:nn { in~first~col } { \Ldots } }
5195   {
5196     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5197     { \@@_error:nn { in~last~col } { \Ldots } }
5198     {
5199       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5200       { #1 , down = #2 , up = #3 , middle = #4 }
5201     }
5202   }
5203   \bool_if:NF \l_@@_nullify_dots_bool
5204   { \phantom { \ensuremath { \@@_old_ldots: } } }
5205   \bool_gset_true:N \g_@@_empty_cell_bool
5206 }
```

```
5207 \cs_new_protected:Npn \@@_Cdots:
5208 { \@@_collect_options:n { \@@_Cdots_i } }
5209 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5210 {
5211   \int_if_zero:nTF { \c@jCol }
5212   { \@@_error:nn { in~first~col } { \Cdots } }
5213   {
5214     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5215     { \@@_error:nn { in~last~col } { \Cdots } }
5216     {
5217       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5218       { #1 , down = #2 , up = #3 , middle = #4 }
5219     }
5220   }
5221   \bool_if:NF \l_@@_nullify_dots_bool
5222   { \phantom { \ensuremath { \@@_old_cdots: } } }
5223   \bool_gset_true:N \g_@@_empty_cell_bool
5224 }
```

```
5225 \cs_new_protected:Npn \@@_Vdots:
5226 { \@@_collect_options:n { \@@_Vdots_i } }
5227 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5228 {
5229   \int_if_zero:nTF { \c@iRow }
```

```

5230 { \@@_error:nn { in~first~row } { \Vdots } }
5231 {
5232   \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5233   { \@@_error:nn { in~last~row } { \Vdots } }
5234   {
5235     \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5236     { #1 , down = #2 , up = #3 , middle = #4 }
5237   }
5238 }
5239 \bool_if:NF \l_@@_nullify_dots_bool
5240 { \phantom { \ensuremath { \@@_old_vdots: } } }
5241 \bool_gset_true:N \g_@@_empty_cell_bool
5242 }

5243 \cs_new_protected:Npn \@@_Ddots:
5244 { \@@_collect_options:n { \@@_Ddots_i } }
5245 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5246 {
5247   \int_case:nnF \c@iRow
5248   {
5249     0 { \@@_error:nn { in~first~row } { \Ddots } }
5250     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5251   }
5252   {
5253     \int_case:nnF \c@jCol
5254     {
5255       0 { \@@_error:nn { in~first~col } { \Ddots } }
5256       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5257     }
5258     {
5259       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5260       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5261       { #1 , down = #2 , up = #3 , middle = #4 }
5262     }
5263   }
5264 }
5265 \bool_if:NF \l_@@_nullify_dots_bool
5266 { \phantom { \ensuremath { \@@_old_ddots: } } }
5267 \bool_gset_true:N \g_@@_empty_cell_bool
5268 }

5269 \cs_new_protected:Npn \@@_Iddots:
5270 { \@@_collect_options:n { \@@_Iddots_i } }
5271 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5272 {
5273   \int_case:nnF \c@iRow
5274   {
5275     0 { \@@_error:nn { in~first~row } { \Iddots } }
5276     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5277   }
5278   {
5279     \int_case:nnF \c@jCol
5280     {
5281       0 { \@@_error:nn { in~first~col } { \Iddots } }
5282       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5283     }
5284     {
5285       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5286       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5287       { #1 , down = #2 , up = #3 , middle = #4 }
5288     }
5289   }

```

```

5290     \bool_if:NF \l_@@_nullify_dots_bool
5291     { \phantom { \ensuremath { \@@_old_iddots: } } }
5292     \bool_gset_true:N \g_@@_empty_cell_bool
5293   }
5294 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5295 \keys_define:nn { nicematrix / Ddots }
5296 {
5297   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5298   draw-first .default:n = true ,
5299   draw-first .value_forbidden:n = true
5300 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5301 \cs_new_protected:Npn \@@_Hspace:
5302 {
5303   \bool_gset_true:N \g_@@_empty_cell_bool
5304   \hspace
5305 }

```

In the environments of `nicematrix`, the command `\multicolumn` is redefined. We will patch the environment `{tabular}` to go back to the previous value of `\multicolumn`.

```

5306 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. TikZ nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5307 \cs_new:Npn \@@_Hdotsfor:
5308 {
5309   \bool_lazy_and:nnTF
5310     { \int_if_zero_p:n { \c@jCol } }
5311     { \int_if_zero_p:n { \l_@@_first_col_int } }
5312   {
5313     \bool_if:NTF \g_@@_after_col_zero_bool
5314     {
5315       \multicolumn { 1 } { c } { }
5316       \@@_Hdotsfor_i:
5317     }
5318     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5319   }
5320   {
5321     \multicolumn { 1 } { c } { }
5322     \@@_Hdotsfor_i:
5323   }
5324 }

```

The command `\@@_Hdotsfor_i:` is defined with `\NewDocumentCommand` because it has an optional argument. Note that such a command defined by `\NewDocumentCommand` is protected and that's why we have put the `\multicolumn` before (in the definition of `\@@_Hdotsfor:`).

```

5325 \hook_gput_code:nnn { begindocument } { . }
5326 {

```

We don't put `!` before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5327   \cs_new_protected:Npn \@@_Hdotsfor_i:
5328   { \@@_collect_options:n { \@@_Hdotsfor_ii } }

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5329 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5330 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5331 {
5332   \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5333   {
5334     \@@_Hdotsfor:nnnn
5335     { \int_use:N \c@iRow }
5336     { \int_use:N \c@jCol }
5337     { #2 }
5338     {
5339       #1 , #3 ,
5340       down = \exp_not:n { #4 } ,
5341       up = \exp_not:n { #5 } ,
5342       middle = \exp_not:n { #6 }
5343     }
5344   }
5345   \prg_replicate:nn { #2 - 1 }
5346   {
5347     &
5348     \multicolumn { 1 } { c } { }
5349     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5350   }
5351 }
5352 }

```

```

5353 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5354 {
5355   \bool_set_false:N \l_@@_initial_open_bool
5356   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5357 \int_set:Nn \l_@@_initial_i_int { #1 }
5358 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5359 \int_compare:nNnTF { #2 } = { \c_one_int }
5360 {
5361   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5362   \bool_set_true:N \l_@@_initial_open_bool
5363 }
5364 {
5365   \cs_if_exist:cTF
5366   {
5367     pgf @ sh @ ns @ \@@_env:
5368     - \int_use:N \l_@@_initial_i_int
5369     - \int_eval:n { #2 - 1 }
5370   }
5371   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5372   {
5373     \int_set:Nn \l_@@_initial_j_int { #2 }
5374     \bool_set_true:N \l_@@_initial_open_bool
5375   }
5376 }
5377 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5378 {
5379   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5380   \bool_set_true:N \l_@@_final_open_bool
5381 }
5382 {
5383   \cs_if_exist:cTF
5384   {
5385     pgf @ sh @ ns @ \@@_env:

```

```

5386         - \int_use:N \l_@@_final_i_int
5387         - \int_eval:n { #2 + #3 }
5388     }
5389     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5390     {
5391         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5392         \bool_set_true:N \l_@@_final_open_bool
5393     }
5394 }
5395 \bool_if:NT \g_@@_aux_found_bool
5396 {
5397     \group_begin:
5398     \@@_open_shorten:
5399     \int_if_zero:nTF { #1 }
5400     { \color { nicematrix-first-row } }
5401     {
5402         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5403         { \color { nicematrix-last-row } }
5404     }
5405     \keys_set:nn { nicematrix / xdots } { #4 }
5406     \@@_color:o \l_@@_xdots_color_tl
5407     \@@_actually_draw_Ldots:
5408     \group_end:
5409 }

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5410 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5411 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5412 }

```

```

5413 \hook_gput_code:nnn { begindocument } { . }
5414 {
5415     \cs_new_protected:Npn \@@_Vdotsfor:
5416     { \@@_collect_options:n { \@@_Vdotsfor_i } }

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that’s why we are in a `\AtBeginDocument`).

```

5417 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5418 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5419 {
5420     \bool_gset_true:N \g_@@_empty_cell_bool
5421     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5422     {
5423         \@@_Vdotsfor:nnnn
5424         { \int_use:N \c_iRow }
5425         { \int_use:N \c_jCol }
5426         { #2 }
5427         {
5428             #1 , #3 ,
5429             down = \exp_not:n { #4 } ,
5430             up = \exp_not:n { #5 } ,
5431             middle = \exp_not:n { #6 }
5432         }
5433     }
5434 }
5435 }

```

**#1** is the number of row;

**#2** is the number of column;

#3 is the numbers of rows which are involved;

```

5436 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5437 {
5438   \bool_set_false:N \l_@@_initial_open_bool
5439   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5440   \int_set:Nn \l_@@_initial_j_int { #2 }
5441   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5442   \int_compare:nNnTF { #1 } = { \c_one_int }
5443   {
5444     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5445     \bool_set_true:N \l_@@_initial_open_bool
5446   }
5447   {
5448     \cs_if_exist:cTF
5449     {
5450       pgf @ sh @ ns @ \@@_env:
5451       - \int_eval:n { #1 - 1 }
5452       - \int_use:N \l_@@_initial_j_int
5453     }
5454     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5455     {
5456       \int_set:Nn \l_@@_initial_i_int { #1 }
5457       \bool_set_true:N \l_@@_initial_open_bool
5458     }
5459   }
5460   \int_compare:nNnTF { #1 + #3 - 1 } = { \c_iRow }
5461   {
5462     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5463     \bool_set_true:N \l_@@_final_open_bool
5464   }
5465   {
5466     \cs_if_exist:cTF
5467     {
5468       pgf @ sh @ ns @ \@@_env:
5469       - \int_eval:n { #1 + #3 }
5470       - \int_use:N \l_@@_final_j_int
5471     }
5472     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5473     {
5474       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5475       \bool_set_true:N \l_@@_final_open_bool
5476     }
5477   }
5478   \bool_if:NT \g_@@_aux_found_bool
5479   {
5480     \group_begin:
5481     \@@_open_shorten:
5482     \int_if_zero:nTF { #2 }
5483     { \color { nicematrix-first-col } }
5484     {
5485       \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5486       { \color { nicematrix-last-col } }
5487     }
5488     \keys_set:nn { nicematrix / xdots } { #4 }
5489     \@@_color:o \l_@@_xdots_color_tl
5490     \bool_if:NTF \l_@@_Vbrace_bool
5491     { \@@_actually_draw_Vbrace: }
5492     { \@@_actually_draw_Vdots: }
5493     \group_end:
5494   }

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5495 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5496 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5497 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5498 \NewDocumentCommand \@@_rotate: { 0 { } }
5499 {
5500   \bool_gset_true:N \g_@@_rotate_bool
5501   \keys_set:nn { nicematrix / rotate } { #1 }
5502   \ignorespaces
5503 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5504 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5505 \keys_define:nn { nicematrix / rotate }
5506 {
5507   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5508   c .value_forbidden:n = true ,
5509   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5510 }

```

## 19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).<sup>14</sup>

```

5511 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5512 {
5513   \tl_if_empty:nTF { #2 }
5514   { #1 }
5515   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5516 }
5517 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5518 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

---

<sup>14</sup>Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.



With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```
5519 \hook_gput_code:nnn { begindocument } { . }
5520 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
5521 \tl_set_rescan:Nnn \l_tmpa_tl { }
5522 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5523 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5524 {
5525   \group_begin:
5526   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5527   \@@_color:o \l_@@_xdots_color_tl
5528   \use:e
5529   {
5530     \@@_line_i:nn
5531     { \@@_double_int_eval:n #2 - \q_stop }
5532     { \@@_double_int_eval:n #3 - \q_stop }
5533   }
5534   \group_end:
5535 }
5536 }

5537 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5538 {
5539   \bool_set_false:N \l_@@_initial_open_bool
5540   \bool_set_false:N \l_@@_final_open_bool
5541   \bool_lazy_or:nnTF
5542   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5543   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5544   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```
5545 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5546 }

5547 \hook_gput_code:nnn { begindocument } { . }
5548 {
5549   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5550   {
```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:`.

```
5551 \c_@@_pgfortikzpicture_tl
5552 \@@_draw_line_iii:nn { #1 } { #2 }
5553 \c_@@_endpgfortikzpicture_tl
5554 }
5555 }
```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```
5556 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5557 {
5558   \pgfrememberpicturepositiononpagetrue
5559   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5560   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5561   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5562   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5563   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5564   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5565   \@@_draw_line:
5566 }
```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `@@_if_row_less_than:nn` is *not* protected.

#1 is the first row *after* the scope of the instructions in #2

However, both arguments are implicit because they are taken by curryfication.

```
5567 \cs_new:Npn @@_if_row_less_than:nn { \int_compare:nNt { \c@iRow } < }
5568 \cs_new:Npn @@_if_col_greater_than:nn { \int_compare:nNf { \c@jCol } < }
```

`@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5569 \cs_set_protected:Npn @@_put_in_row_style:n #1
5570 {
5571   \tl_gput_right:Ne \g_@@_row_style_tl
5572   {
```

Be careful, `\exp_not:N @@_if_row_less_than:nn` can't be replaced by a protected version of `@@_if_row_less_than:nn`.

```
5573   \exp_not:N
5574   @@_if_row_less_than:nn
5575   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```
5576   {
5577     \exp_not:N
5578     @@_if_col_greater_than:nn
5579     { \int_eval:n { \c@jCol } }
5580     { \exp_not:n { #1 } \scan_stop: }
5581   }
5582 }
5583 }
5584 \cs_generate_variant:Nn @@_put_in_row_style:n { e }
```

```
5585 \keys_define:nn { nicematrix / RowStyle }
5586 {
5587   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5588   cell-space-top-limit .value_required:n = true ,
5589   cell-space-top-limit+ .code:n =
5590     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5591   cell-space-top-limit+ .value_required:n = true ,
5592   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5593   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5594   cell-space-bottom-limit .value_required:n = true ,
5595   cell-space-bottom-limit+ .code:n =
5596     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5597   cell-space-bottom-limit+ .value_required:n = true ,
5598   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5599   cell-space-limits .meta:n =
```

```

5600     {
5601         cell-space-top-limit = #1 ,
5602         cell-space-bottom-limit = #1 ,
5603     } ,
5604     cell-space-limits+ .meta:n =
5605     {
5606         cell-space-top-limit += #1 ,
5607         cell-space-bottom-limit += #1 ,
5608     } ,
5609     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5610     color .tl_set:N = \l_@@_color_tl ,
5611     color .value_required:n = true ,
5612     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5613     bold .default:n = true ,
5614     nb-rows .code:n =
5615         \str_if_eq:eeTF { #1 } { * }
5616         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5617         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5618     nb-rows .value_required:n = true ,
5619     fill .tl_set:N = \l_@@_fill_tl ,
5620     fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

5621     opacity .tl_set:N = \l_@@_opacity_tl ,
5622     opacity .value_required:n = true ,
5623     rowcolor .tl_set:N = \l_@@_fill_tl ,
5624     rowcolor .value_required:n = true ,
5625     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5626     rounded-corners .default:n = 4 pt ,
5627     unknown .code:n =
5628         \@@_unknown_key:nn
5629         { nicematrix / RowStyle }
5630         { Unknown-key-for-RowStyle }
5631 }

```

```

5632 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5633 {
5634     \group_begin:
5635     \tl_clear:N \l_@@_fill_tl
5636     \tl_clear:N \l_@@_opacity_tl
5637     \tl_clear:N \l_@@_color_tl
5638     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5639     \dim_zero:N \l_@@_rounded_corners_dim
5640     \dim_zero:N \l_tmpa_dim
5641     \dim_zero:N \l_tmpb_dim
5642     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `fill` (or its alias `rowcolor`) has been used.

```

5643     \tl_if_empty:NF \l_@@_fill_tl
5644     {
5645         \@@_add_opacity_to_fill:
5646         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5647         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5648         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5649         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5650         {
5651             \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5652             - *
5653         }
5654         { \dim_use:N \l_@@_rounded_corners_dim }
5655     }
5656 }

```

```

5657 \@@_put_in_row_style:n { \exp_not:n { #2 } }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5658 \dim_compare:nNtT { \l_tmpa_dim } > { \c_zero_dim }
5659 {
5660 \@@_put_in_row_style:e
5661 {
5662 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5663 {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5664 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5665 { \dim_use:N \l_tmpa_dim }
5666 }
5667 }
5668 }

```

`\l_tmpb_dim` is the value of the key cell-space-bottom-limit of `\RowStyle`.

```

5669 \dim_compare:nNtT { \l_tmpb_dim } > { \c_zero_dim }
5670 {
5671 \@@_put_in_row_style:e
5672 {
5673 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5674 {
5675 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5676 { \dim_use:N \l_tmpb_dim }
5677 }
5678 }
5679 }

```

`\l_@@_color_tl` is the value of the key color of `\RowStyle`.

```

5680 \tl_if_empty:NF \l_@@_color_tl
5681 {
5682 \@@_put_in_row_style:e
5683 {
5684 \mode_leave_vertical:
5685 \@@_color:n { \l_@@_color_tl }
5686 }
5687 }

```

`\l_@@_bold_row_style_bool` is the value of the key bold.

```

5688 \bool_if:NT \l_@@_bold_row_style_bool
5689 {
5690 \@@_put_in_row_style:n
5691 {
5692 \exp_not:n
5693 {
5694 \if_mode_math:
5695 $ % $
5696 \bfseries \boldmath
5697 $ % $
5698 \else:
5699 \bfseries \boldmath
5700 \fi:
5701 }
5702 }
5703 }
5704 \group_end:
5705 \g_@@_row_style_tl
5706 \ignorespaces
5707 }

```

The following commande must *not* be protected.

```

5708 \cs_new:Npn \@@_rounded_from_row:n #1
5709 {
5710 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5711 { \int_eval:n { #1 } - 1 }
5712 {
5713   \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5714   - \exp_not:n { \int_use:N \c@jCol }
5715 }
5716 { \dim_use:N \l_@@_rounded_corners_dim }
5717 }

```

## 21 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

`#1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn’t only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```

5718 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5719 {

```

First, we look for the number of the color and, if it’s found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```

5720   \int_zero:N \l_tmpa_int

```

We don’t take into account the colors like `myserie!!+` because those colors are special color from a `\definecolorseries` of `xcolor`. `\str_if_in:nnF` is mandatory: don’t use `\tl_if_in:nnF`.

```

5721   \str_if_in:nnF { #1 } { !! }
5722   {
5723     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5724     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5725   }
5726   \int_if_zero:nTF { \l_tmpa_int }

```

First, the case where the color is a *new* color (not in the sequence).

```

5727   {
5728     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5729     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5730   }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5731     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5732   }
5733   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5734   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

The following command must be used within a `\pgfpicture`.

```

5735   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5736   {
5737     \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5738     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5739     \group_begin:
5740     \pgfsetcornersarced
5741     {
5742       \pgfpoint
5743       { \l_@@_tab_rounded_corners_dim }
5744       { \l_@@_tab_rounded_corners_dim }
5745     }

```

Because we want `nicematrix` compatible with arrays constructed by `array`, the nodes for the rows and columns (that is to say the nodes `row-i` and `col-j`) have not always the expected position, that is to say, there is sometimes a slight shifting of something such as `\arrayrulewidth`. Now, for the clipping, we have to change slightly the position of that clipping whether a rounded rectangle around the array is required. That's the point which is tested in the following line.

```

5746     \bool_if:NTF \l_@@_hvlines_bool
5747     {
5748       \pgfpathrectanglecorners
5749       {
5750         \pgfpointadd
5751         { \@@_qpoint:n { row-1 } }
5752         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5753       }
5754       {
5755         \pgfpointadd
5756         {
5757           \@@_qpoint:n
5758           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5759         }
5760         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5761       }
5762     }
5763     {
5764       \pgfpathrectanglecorners
5765       { \@@_qpoint:n { row-1 } }
5766       {
5767         \pgfpointadd
5768         {
5769           \@@_qpoint:n
5770           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5771         }
5772         { \pgfpoint \c_zero_dim \arrayrulewidth }
5773       }
5774     }
5775     \pgfusepath { clip }
5776     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5777   }
5778 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5779 \cs_new_protected:Npn \@@_actually_color:
5780 {
5781   \pgfpicture
5782   \pgf@relevantforpicturesizefalse

```

If the final user has used the key `rounded-corners` for the environment `{NiceTabular}`, we will clip to a rectangle with rounded corners before filling the rectangles.

```

5783   \@@_clip_with_rounded_corners:
5784   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5785   {
5786     \int_compare:nNnTF { ##1 } = { \c_one_int }
5787     {
5788       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5789       \use:c { g_@@_color _ 1 _tl }
5790       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5791     }
5792     {
5793       \begin { pgfscope }
5794         \@@_color_opacity: ##2
5795         \use:c { g_@@_color _ ##1 _tl }
5796         \tl_gclear:c { g_@@_color _ ##1 _tl }
5797         \pgfusepath { fill }
5798       \end { pgfscope }
5799     }
5800   }
5801   \endpgfpicture
5802 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5803 \cs_new_protected:Npn \@@_color_opacity:
5804 {
5805   \peek_meaning:NTF [
5806     { \@@_color_opacity:w }
5807     { \@@_color_opacity:w [ ] }
5808   }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5809 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5810 {
5811   \tl_clear:N \l_tmpa_tl
5812   \keys_set_known:nn { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5813   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5814   \tl_if_empty:NTF \l_tmpb_tl
5815     { \@declaredcolor }
5816     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5817 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5818 \keys_define:nn { nicematrix / color-opacity }
5819 {
5820   opacity .tl_set:N          = \l_tmpa_tl ,
5821   opacity .value_required:n = true
5822 }

```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5823 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5824 {
5825   \def \l_@@_rows_tl { #1 }
5826   \def \l_@@_cols_tl { #2 }
5827   \@@_cartesian_path:
5828 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5829 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5830 {
5831   \tl_if_blank:nF { #2 }
5832   {
5833     \@@_add_to_colors_seq:en
5834     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5835     { \@@_cartesian_color:nn { #3 } { - } }
5836   }
5837 }

```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5838 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5839 {
5840   \tl_if_blank:nF { #2 }
5841   {
5842     \@@_add_to_colors_seq:en
5843     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5844     { \@@_cartesian_color:nn { - } { #3 } }
5845   }
5846 }

```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5847 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5848 {
5849   \tl_if_blank:nF { #2 }
5850   {
5851     \@@_add_to_colors_seq:en
5852     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5853     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5854   }
5855 }

```

The last argument is the radius of the corners of the rectangle.

```

5856 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5857 {
5858   \tl_if_blank:nF { #2 }
5859   {
5860     \@@_add_to_colors_seq:en
5861     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5862     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5863   }
5864 }

```

The last argument is the radius of the corners of the rectangle.

```

5865 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5866 {
5867   \@@_cut_on_hyphen:w #1 \q_stop
5868   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5869   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5870   \@@_cut_on_hyphen:w #2 \q_stop
5871   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5872   \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```



The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
5873 \@@_cartesian_path:n { #3 }
5874 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
5875 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5876 {
5877   \clist_map_inline:nn { #3 }
5878   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5879 }
```

```
5880 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5881 {
5882   \int_step_inline:nn { \c@iRow }
5883   {
5884     \int_step_inline:nn { \c@jCol }
5885     {
5886       \int_if_even:nTF { ####1 + ##1 }
5887       { \@@_cellcolor [ #1 ] { #2 } }
5888       { \@@_cellcolor [ #1 ] { #3 } }
5889       { ##1 - ####1 }
5890     }
5891   }
5892 }
```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```
5893 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5894 {
5895   \@@_rectanglecolor [ #1 ] { #2 }
5896   { 1 - 1 }
5897   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5898 }
```

```
5899 \keys_define:nn { nicematrix / rowcolors }
5900 {
5901   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5902   respect-blocks .default:n = true ,
5903   cols .tl_set:N = \l_@@_cols_tl ,
5904   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5905   restart .default:n = true ,
5906   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5907 }
```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

`#1` (optional) is the color space; `#2` is a list of intervals of rows; `#3` is the list of colors; `#4` is for the optional list of pairs `key=value`.

```
5908 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5909 {
```

`\l_@@_colors_seq` will be the list of colors.

```

5910 \pgfpicture
5911 \pgf@relevantforpicturesizefalse
5912 \seq_clear_new:N \l_@@_colors_seq
5913 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5914 \tl_clear_new:N \l_@@_cols_tl
5915 \tl_set:Nn \l_@@_cols_tl { - }
5916 \keys_set:nn { nicematrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

5917 \int_zero_new:N \l_@@_color_int
5918 \int_set_eq:NN \l_@@_color_int \c_one_int
5919 \bool_if:NT \l_@@_respect_blocks_bool
5920 {

```

We don't want to take into account a block which is entirely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5921 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5922 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5923 { \@@_not_in_exterior_p:nnnnn ##1 }
5924 }

```

`#2` is the list of intervals of rows.

```

5925 \clist_map_inline:nn { #2 }
5926 {
5927 \tl_set:Nn \l_tmpa_tl { ##1 }
5928 \tl_if_in:NnTF \l_tmpa_tl { - }
5929 { \@@_cut_on_hyphen:w ##1 \q_stop }
5930 { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5931 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5932 \int_set:Nn \l_@@_color_int
5933 { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5934 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5935 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5936 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5937 \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5938 \bool_if:NT \l_@@_respect_blocks_bool
5939 {
5940 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5941 { \@@_intersect_our_row_p:nnnnn #####1 }
5942 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5943 }
5944 \tl_set:Nn \l_@@_rows_tl
5945 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5946 \tl_set:Nn \l_@@_color_tl
5947 {
5948 \@@_color_index:n
5949 {
5950 \int_mod:nn
5951 { \l_@@_color_int - 1 }
5952 { \seq_count:N \l_@@_colors_seq }
5953 + 1
5954 }
5955 }

```

```

5956         \tl_if_empty:NF \l_@@_color_tl
5957         {
5958             \@@_add_to_colors_seq:ee
5959             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5960             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5961         }
5962         \int_incr:N \l_@@_color_int
5963         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5964     }
5965 }
5966 \endpgfpicture
5967 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index #1. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

5968 \cs_new:Npn \@@_color_index:n #1
5969 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5970     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5971     { \@@_color_index:n { #1 - 1 } }
5972     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5973 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5974 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5975 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5976 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5977 {
5978     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5979     { \int_set:Nn \l_tmpb_int { #3 } }
5980 }

5981 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5982 {
5983     \int_if_zero:nTF { #4 }
5984     { \prg_return_false: }
5985     {
5986         \int_compare:nNnTF { #2 } > { \c@jCol }
5987         { \prg_return_false: }
5988         { \prg_return_true: }
5989     }
5990 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5991 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5992 {
5993     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5994     { \prg_return_false: }
5995     {
5996         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5997         { \prg_return_false: }
5998         { \prg_return_true: }
5999     }
6000 }

```

The following command uses two implicit arguments: `\l_@@_rows_tl` and `\l_@@_cols_tl` which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command `\@@_cartesian_path:` which corresponds to a value 0 pt for the radius of the corners. This command is, in particular, used in `\@@_rectanglecolor:nnn` (used in `\@@_rectanglecolor`, itself used in `\@@_cellcolor`).

```

6001 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6002 {
6003   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
6004   {
6005     \bool_if:NTF \l_@@_nocolor_used_bool
6006     { \@@_cartesian_path_normal_ii: }
6007     {
6008       \clist_if_empty:NTF \l_@@_corners_cells_clist
6009       { \@@_cartesian_path_normal_i:n { #1 } }
6010       { \@@_cartesian_path_normal_ii: }
6011     }
6012   }
6013   { \@@_cartesian_path_normal_i:n { #1 } }
6014 }

```

First, the situation where is a rectangular zone of cells will be colored as a whole (in the instructions of the resulting PDF). The argument is the radius of the corners.

```

6015 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6016 {
6017   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6018   \clist_map_inline:Nn \l_@@_cols_tl
6019   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6020     \def \l_tmpa_tl { ##1 }
6021     \tl_if_in:NnTF \l_tmpa_tl { - }
6022     { \@@_cut_on_hyphen:w ##1 \q_stop }
6023     { \def \l_tmpb_tl { ##1 } }
6024     \tl_if_empty:NTF \l_tmpa_tl
6025     { \def \l_tmpa_tl { 1 } }
6026     {
6027       \str_if_eq:eeT { \l_tmpa_tl } { * }
6028       { \def \l_tmpa_tl { 1 } }
6029     }
6030     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
6031     { \@@_error:n { Invalid~col~number } }
6032     \tl_if_empty:NTF \l_tmpb_tl
6033     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6034     {
6035       \str_if_eq:eeT { \l_tmpb_tl } { * }
6036       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6037     }
6038     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
6039     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6040     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6041     \@@_qpoint:n { col - \l_tmpa_tl }
6042     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
6043     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6044     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6045     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6046     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows. We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6047 \clist_map_inline:Nn \l_@@_rows_tl
6048 {
6049   \def \l_tmpa_tl { #####1 }
6050   \tl_if_in:NnTF \l_tmpa_tl { - }
6051     { \@@_cut_on_hyphen:w #####1 \q_stop }
6052     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6053   \tl_if_empty:NTF \l_tmpa_tl
6054     { \def \l_tmpa_tl { 1 } }
6055     {
6056       \str_if_eq:eeT { \l_tmpa_tl } { * }
6057       { \def \l_tmpa_tl { 1 } }
6058     }
6059   \tl_if_empty:NTF \l_tmpb_tl
6060     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6061     {
6062       \str_if_eq:eeT { \l_tmpb_tl } { * }
6063       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6064     }
6065   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
6066     { \@@_error:n { Invalid-row-number } }
6067   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
6068     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

6069 \cs_if_exist:cF
6070 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6071 {
6072   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6073   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6074   \@@_qpoint:n { row - \l_tmpa_tl }
6075   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6076   \pgfpathrectanglecorners
6077     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6078     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6079 }
6080 }
6081 }
6082 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6083 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6084 {
6085   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6086   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6087 \clist_map_inline:Nn \l_@@_cols_tl
6088 {
6089   \@@_qpoint:n { col - ##1 }
6090   \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
6091     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6092     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6093   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6094   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6095 \clist_map_inline:Nn \l_@@_rows_tl
6096 {
6097   \@@_if_in_corner:nF { #####1 - ##1 }
6098   {
6099     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6100     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6101     \@@_qpoint:n { row - #####1 }

```

```

6102         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6103         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6104         {
6105             \pgfpathrectanglecorners
6106             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6107             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6108         }
6109     }
6110 }
6111 }
6112 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor` and `\@@_rowcolor:n` (used in `\@@_rowcolor`).

```

6113 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n \c_zero_dim }

```

Despite its name, the following command does not create a PGF path. It declares as colored by the “empty color” all the cells in what would be the path. Hence, the other coloring instructions of `nicematrix` won’t put color in those cells. the

```

6114 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6115 {
6116     \bool_set_true:N \l_@@_nocolor_used_bool
6117     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6118     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6119     \clist_map_inline:Nn \l_@@_rows_tl
6120     {
6121         \clist_map_inline:Nn \l_@@_cols_tl
6122         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6123     }
6124 }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-\* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

6125 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6126 {
6127     \clist_set_eq:NN \l_tmpa_clist #1
6128     \clist_clear:N #1
6129     \clist_map_inline:Nn \l_tmpa_clist
6130     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6131     \def \l_tmpa_tl { ##1 }
6132     \tl_if_in:NnTF \l_tmpa_tl { - }
6133     { \@@_cut_on_hyphen:w ##1 \q_stop }
6134     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6135     \bool_lazy_or:nnT
6136     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
6137     { \tl_if_blank_p:o \l_tmpa_tl }
6138     { \def \l_tmpa_tl { 1 } }
6139     \bool_lazy_or:nnT
6140     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
6141     { \tl_if_blank_p:o \l_tmpb_tl }
6142     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6143     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6144     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6145     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
6146     { \clist_put_right:Nn #1 { #####1 } }
6147 }
6148 }

```

The following command will be linked to `\cellcolor` in the tabular.

```
6149 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6150 {
6151   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6152   {
```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```
6153     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6154     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6155   }
6156   \ignorespaces
6157 }

6158 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6159 { \@@_error:n { cellcolor~in~Block } }
6160 % \end{macrocode}
6161 %
6162 % \begin{macrocode}
6163 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6164 { \@@_error:n { rowcolor~in~Block } }
6165 % \end{macrocode}
6166 %
6167 % \bigskip
6168 % The following command will be linked to |\rowcolor| in the tabular.
6169 % \begin{macrocode}
6170 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6171 {
6172   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6173   {
6174     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6175     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6176     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6177   }
6178   \ignorespaces
6179 }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
6180 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6181 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
6182 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6183 {
```

A use of `\rowlistcolors` in the tabular erases the instructions `\rowlistcolors` which are in force. However, it's possible to put *several* instructions `\rowlistcolors` in the same row of a tabular: it may be useful when those instructions `\rowlistcolors` concerns different columns of the tabular (thanks to the key `cols` of `\rowlistcolors`). That's why we store the different instructions `\rowlistcolors` which are in force in a sequence `\g_@@_rowlistcolors_seq`. Now, we will filter that sequence to keep only the elements which have been issued on the actual row. We will store the elements to keep in the `\g_tmpa_seq`.

```
6184   \seq_gclear:N \g_tmpa_seq
6185   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6186   { \@@_rowlistcolors_tabular:nnnn #1 }
6187   \seq_gset_eq:NN \g_@@_rowlistcolors_seq \g_tmpa_seq
```

Now, we add to the sequence `\g_@@_rowlistcolors_seq` (which is the list of the commands `\rowlistcolors` which are in force) the current instruction `\rowlistcolors`.

```
6188   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6189   {
```

```

6190     { \int_use:N \c@iRow }
6191     { \exp_not:n { #1 } }
6192     { \exp_not:n { #2 } }
6193     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6194   }
6195   \ignorespaces
6196 }

```

The following command will be applied to each component of `\g_@@_rowlistcolors_seq`. Each component of that sequence is a kind of 4-uple of the form `{#1}{#2}{#3}{#4}`.

**#1** is the number of the row where the command `\rowlistcolors` has been issued.

**#2** is the colorimetric space (optional argument of the `\rowlistcolors`).

**#3** is the list of colors (mandatory argument of `\rowlistcolors`).

**#4** is the list of *key=value* pairs (last optional argument of `\rowlistcolors`).

```

6197 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6198 {
6199   \int_compare:nNnTF { #1 } = { \c@iRow }

```

We (temporary) keep in memory in `\g_tmpa_seq` the instructions which will still be in force after the current instruction (because they have been issued in the same row of the tabular).

```

6200   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6201   {
6202     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6203     {
6204       \@@_rowlistcolors
6205       [ \exp_not:n { #2 } ]
6206       { #1 - \int_eval:n { \c@iRow - 1 } }
6207       { \exp_not:n { #3 } }
6208       [ \exp_not:n { #4 } ]
6209     }
6210   }
6211 }

```

The following command will be used at the end of the tabular, just before the execution of the `\g_@@_pre_code_before_tl`. It clears the sequence `\g_@@_rowlistcolors_seq` of all the commands `\rowlistcolors` which are (still) in force.

```

6212 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6213 {
6214   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6215   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6216   \seq_gclear:N \g_@@_rowlistcolors_seq
6217 }
6218 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6219 {
6220   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6221   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6222 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form *i*: it means that the command must be applied to all the rows from the row *i* until the end of the tabular.

```

6223 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6224 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

6225   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6226   {

```



You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```

6227         \tl_gput_left:N\g_@@_pre_code_before_tl
6228         {
6229             \exp_not:N \columncolor [ #1 ]
6230             { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6231         }
6232     }
6233 }

6234 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6235 {
6236     \clist_map_inline:nn { #1 }
6237     {
6238         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6239         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6240         \columncolor { nocolor } { ##1 }
6241     }
6242 }

6243 \cs_new_protected:Npn \@@_EmptyRow:n #1
6244 {
6245     \clist_map_inline:nn { #1 }
6246     {
6247         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6248         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6249         \rowcolor { nocolor } { ##1 }
6250     }
6251 }

```

## 22 The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

6252 \cs_new_eq:NN \OnlyMainNiceMatrix \use:n

```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```

6253 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6254 {
6255     \int_if_zero:nTF { \l_@@_first_col_int }
6256     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6257     {
6258         \int_if_zero:nTF { \c@jCol }
6259         {
6260             \int_compare:nNnF { \c@iRow } = { -1 }
6261             {

```

```

6262         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6263         { #1 }
6264     }
6265 }
6266 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6267 }
6268 }

```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell. The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```

6269 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6270 {
6271     \int_if_zero:nF { \c@iRow }
6272     {
6273         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6274         {
6275             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6276             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6277         }
6278     }
6279 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to  $-2$  or  $-1$  (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6280 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6281 {
6282     \IfPackageLoadedTF { tikz }
6283     {
6284         \IfPackageLoadedTF { booktabs }
6285         { #2 }
6286         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6287     }
6288     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6289 }
6290 \NewExpandableDocumentCommand { \@@_TopRule } { }
6291 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6292 \cs_new:Npn \@@_TopRule_i:
6293 {
6294     \noalign \bgroup
6295     \peek_meaning:NTF [
6296     { \@@_TopRule_ii: }
6297     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6298 }
6299 \NewDocumentCommand \@@_TopRule_ii: { o }
6300 {
6301     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6302     {
6303         \@@_hline:n
6304         {
6305             position = \int_eval:n { \c@iRow + 1 } ,
6306             tikz =
6307             {
6308                 line-width = #1 ,
6309                 yshift = 0.25 \arrayrulewidth ,
6310                 shorten~< = - 0.5 \arrayrulewidth
6311             } ,
6312             total-width = #1
6313         }

```

```

6314     }
6315     \skip_vertical:n { \belowrulesep + #1 }
6316     \egroup
6317 }

6318 \NewExpandableDocumentCommand { \@@_BottomRule } { } {
6319   { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6320   \cs_new:Npn \@@_BottomRule_i:
6321     {
6322       \noalign \bgroup
6323       \peek_meaning:NTF [
6324         { \@@_BottomRule_ii: }
6325         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6326       }

6327   \NewDocumentCommand \@@_BottomRule_ii: { o }
6328   {
6329     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6330     {
6331       \@@_hline:n
6332       {
6333         position = \int_eval:n { \c@iRow + 1 } ,
6334         tikz =
6335           {
6336             line-width = #1 ,
6337             yshift = 0.25 \arrayrulewidth ,
6338             shorten-< = - 0.5 \arrayrulewidth
6339           } ,
6340         total-width = #1 ,
6341       }
6342     }
6343     \skip_vertical:N \aboverulesep
6344     \@@_create_row_node_i:
6345     \skip_vertical:n { #1 }
6346     \egroup
6347   }

6348 \NewExpandableDocumentCommand { \@@_MidRule } { } {
6349   { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6350   \cs_new:Npn \@@_MidRule_i:
6351     {
6352       \noalign \bgroup
6353       \peek_meaning:NTF [
6354         { \@@_MidRule_ii: }
6355         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6356       }

6357   \NewDocumentCommand \@@_MidRule_ii: { o }
6358   {
6359     \skip_vertical:N \aboverulesep
6360     \@@_create_row_node_i:
6361     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6362     {
6363       \@@_hline:n
6364       {
6365         position = \int_eval:n { \c@iRow + 1 } ,
6366         tikz =
6367           {
6368             line-width = #1 ,
6369             yshift = 0.25 \arrayrulewidth ,
6370             shorten-< = - 0.5 \arrayrulewidth
6371           } ,
6372         total-width = #1 ,
6373       }
6374     }

```

```

6375 \skip_vertical:n { \belowrulesep + #1 }
6376 \egroup
6377 }

```

## General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of `key=value` pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```

6378 \keys_define:nn { nicematrix / Rules }
6379 {
6380   position .int_set:N = \l_@@_position_int ,
6381   position .value_required:n = true ,
6382   start .int_set:N = \l_@@_start_int ,
6383   end .code:n =
6384     \bool_lazy_or:nnTF
6385     { \tl_if_empty_p:n { #1 } }
6386     { \str_if_eq_p:ee { #1 } { last } }
6387     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6388     { \int_set:Nn \l_@@_end_int { #1 } }
6389 }

```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analysis is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6390 \keys_define:nn { nicematrix / RulesBis }
6391 {
6392   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6393   multiplicity .initial:n = 1 ,
6394   dotted .bool_set:N = \l_@@_dotted_bool ,
6395   dotted .initial:n = false ,
6396   dotted .default:n = true ,

```

We want that, even when the rule has been defined with TikZ by the key `tikz`, the user has still the possibility to change the color of the rule with the key `color` (in the command `\Hline`, not in the key `tikz` of the command `\Hline`). The main use is, when the user has defined its own command `\MyDashedLine` by `\newcommand{\MyDashedRule}{\Hline[tikz=dashed]}`, to give the ability to write `\MyDashedRule[color=red]`.

```

6397   color .code:n =
6398     \@@_set_CTarc:n { #1 }
6399     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6400   color .value_required:n = true ,
6401   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6402   sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with TikZ.

```

6403   tikz .code:n =
6404     \IfPackageLoadedTF { tikz }
6405     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6406     { \@@_error:n { tikz-without-tikz } } ,
6407   tikz .value_required:n = true ,
6408   total-width .dim_set:N = \l_@@_rule_width_dim ,
6409   total-width .value_required:n = true ,
6410   width .meta:n = { total-width = #1 } ,
6411   unknown .code:n =
6412     \@@_unknown_key:nn

```

```

6413         { nicematrix / RulesBis }
6414         { Unknown-key-for-RulesBis }
6415     }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```

6416 \cs_new_protected:Npn \@@_vline:n #1
6417 {

```

The group is for the options.

```

6418     \group_begin:
6419     \int_set_eq:NN \l_@@_end_int \c@iRow
6420     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6421     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6422     \@@_vline_i:
6423     \group_end:
6424 }

```

```

6425 \cs_new_protected:Npn \@@_vline_i:
6426 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

6427     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6428     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6429     \l_tmpa_tl
6430     {

```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small vertical rule won't be drawn.

```

6431         \bool_gset_true:N \g_tmpa_bool
6432         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6433         { \@@_test_vline_in_block:nnnnn ##1 }
6434         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6435         { \@@_test_vline_in_block:nnnnn ##1 }
6436         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6437         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6438         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6439         \bool_if:NTF \g_tmpa_bool
6440         {
6441             \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6442             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6443         }
6444         {
6445             \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6446             {
6447                 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6448                 \@@_vline_ii:
6449                 \int_zero:N \l_@@_local_start_int
6450             }
6451         }
6452     }
6453     \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6454     {

```

```

6455     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6456     \@@_vline_ii:
6457   }
6458 }

6459 \cs_new_protected:Npn \@@_test_in_corner_v:
6460 {
6461   \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6462   {
6463     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6464     { \bool_set_false:N \g_tmpa_bool }
6465   }
6466   {
6467     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6468     {
6469       \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6470       { \bool_set_false:N \g_tmpa_bool }
6471       {
6472         \@@_if_in_corner:nT
6473         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6474         { \bool_set_false:N \g_tmpa_bool }
6475       }
6476     }
6477   }
6478 }

6479 \cs_new_protected:Npn \@@_vline_ii:
6480 {
6481   \tl_clear:N \l_@@_tikz_rule_tl
6482   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6483   \bool_if:NTF \l_@@_dotted_bool
6484   { \@@_vline_iv: }
6485   {
6486     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6487     { \@@_vline_iii: }
6488     { \@@_vline_v: }
6489   }
6490 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6491 \cs_new_protected:Npn \@@_vline_iii:
6492 {
6493   \pgfpicture
6494   \pgfrememberpicturepositiononpagetrue
6495   \pgf@relevantforpicturesizefalse
6496   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6497   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6498   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6499   \dim_set:Nn \l_tmpb_dim
6500   {
6501     \pgf@x
6502     - 0.5 \l_@@_rule_width_dim
6503     +
6504     ( \arrayrulewidth * \l_@@_multiplicity_int
6505       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6506   }
6507   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6508   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6509   \bool_lazy_all:nT
6510   {
6511     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }

```

```

6512     { \cs_if_exist_p:N \CT@drsc@ }
6513     { ! \tl_if_blank_p:o \CT@drsc@ }
6514   }
6515   {
6516     \group_begin:
6517     \CT@drsc@
6518     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6519     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6520     \dim_set:Nn \l_@@_tmpd_dim
6521     {
6522       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6523       * ( \l_@@_multiplicity_int - 1 )
6524     }
6525     \pgfpathrectanglecorners
6526     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6527     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6528     \pgfusepath { fill }
6529     \group_end:
6530   }
6531   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6532   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6533   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6534   {
6535     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6536     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6537     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6538   }
6539   \CT@arc@
6540   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6541   \pgfsetrectcap
6542   \pgfusepathqstroke
6543   \endpgfpicture
6544 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6545 \cs_new_protected:Npn \@@_vline_iv:
6546   {
6547     \pgfpicture
6548     \pgfrememberpicturepositiononpagetrue
6549     \pgf@relevantforpicturesizefalse
6550     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6551     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6552     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6553     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6554     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6555     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6556     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6557     \CT@arc@
6558     \@@_draw_line:
6559     \endpgfpicture
6560   }

```

The following code is for the case when the user uses the key `tikz`.

```

6561 \cs_new_protected:Npn \@@_vline_v:
6562   {
6563     \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6564     \CT@arc@
6565     \tl_if_empty:NF \l_@@_rule_color_tl
6566     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }

```

```

6567 \pgfrememberpicturepositiononpagetrue
6568 \pgf@relevantforpicturesizefalse
6569 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6570 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6571 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6572 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6573 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6574 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6575 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6576 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6577 ( \l_tmpb_dim , \l_tmpa_dim ) --
6578 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6579 \end { tikzpicture }
6580 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

6581 \cs_new_protected:Npn \@@_draw_vlines:
6582 {
6583   \int_step_inline:nnn
6584   {
6585     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6586     { 2 }
6587     { 1 }
6588   }
6589   {
6590     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6591     { \c@jCol }
6592     { \int_eval:n { \c@jCol + 1 } }
6593   }
6594   {
6595     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6596     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6597     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6598   }
6599 }

```

## The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{nicematrix/Rules}`.

```

6600 \cs_new_protected:Npn \@@_hline:n #1
6601 {

```

The group is for the options.

```

6602   \group_begin:
6603   \int_set_eq:NN \l_@@_end_int \c@jCol
6604   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6605   \@@_hline_i:
6606   \group_end:
6607 }
6608 \cs_new_protected:Npn \@@_hline_i:
6609 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

6610   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6611   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6612   \l_tmpb_tl
6613   {

```



The boolean `\g_tmpa_bool` indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```
6614 \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6615 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6616 { \@@_test_hline_in_block:nnnnn ##1 }

6617 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6618 { \@@_test_hline_in_block:nnnnn ##1 }

6619 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6620 { \@@_test_hline_in_stroken_block:nnnn ##1 }

6621 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6622 \bool_if:NTF \g_tmpa_bool
6623 {
6624 \int_if_zero:nT { \l_@@_local_start_int }
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
6625 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6626 }
6627 {
6628 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6629 {
6630 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6631 \@@_hline_ii:
6632 \int_zero:N \l_@@_local_start_int
6633 }
6634 }
6635 }
6636 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6637 {
6638 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6639 \@@_hline_ii:
6640 }
6641 }
```

```
6642 \cs_new_protected:Npn \@@_test_in_corner_h:
6643 {
6644 \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6645 {
6646 \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6647 { \bool_set_false:N \g_tmpa_bool }
6648 }
6649 {
6650 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6651 {
6652 \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6653 { \bool_set_false:N \g_tmpa_bool }
6654 {
6655 \@@_if_in_corner:nT
6656 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6657 { \bool_set_false:N \g_tmpa_bool }
6658 }
6659 }
6660 }
6661 }
```

```
6662 \cs_new_protected:Npn \@@_hline_ii:
6663 {
```

```

6664 \tl_clear:N \l_@@_tikz_rule_tl
6665 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6666 \bool_if:NTF \l_@@_dotted_bool
6667 { \@@_hline_iv: }
6668 {
6669   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6670   { \@@_hline_iii: }
6671   { \@@_hline_v: }
6672 }
6673 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6674 \cs_new_protected:Npn \@@_hline_iii:
6675 {
6676   \pgfpicture
6677   \pgfrememberpicturepositiononpagetrue
6678   \pgf@relevantforpicturesizefalse
6679   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6680   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6681   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6682   \dim_set:Nn \l_tmpb_dim
6683   {
6684     \pgf@y
6685     - 0.5 \l_@@_rule_width_dim
6686     +
6687     ( \arrayrulewidth * \l_@@_multiplicity_int
6688       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6689   }
6690   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6691   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6692   \bool_lazy_all:nT
6693   {
6694     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6695     { \cs_if_exist_p:N \CT@drsc@ }
6696     { ! \tl_if_blank_p:o \CT@drsc@ }
6697   }
6698   {
6699     \group_begin:
6700     \CT@drsc@
6701     \dim_set:Nn \l_@@_tmpd_dim
6702     {
6703       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6704       * ( \l_@@_multiplicity_int - 1 )
6705     }
6706     \pgfpathrectanglecorners
6707     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6708     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6709     \pgfusepathqfill
6710     \group_end:
6711   }
6712   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6713   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6714   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6715   {
6716     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6717     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6718     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6719   }
6720   \CT@arc@
6721   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6722   \pgfsetrectcap
6723   \pgfusepathqstroke
6724   \endpgfpicture

```

6725 } }

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6726 \cs_new_protected:Npn \@@_hline_iv:
6727 {
6728   \pgfpicture
6729   \pgfrememberpicturepositiononpagetrue
6730   \pgf@relevantforpicturesizefalse
6731   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6732   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6733   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6734   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6735   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6736   \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6737   {
6738     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6739     \bool_if:NF \g_@@_delims_bool
6740     { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_xdots_inter_dim` is *ad hoc* for a better result.

```
6741   \tl_if_eq:NnF \g_@@_left_delim_tl (
6742     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6743   )
6744   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6745   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6746   \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }
6747   {
6748     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6749     \bool_if:NF \g_@@_delims_bool
6750     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6751     \tl_if_eq:NnF \g_@@_right_delim_tl )
6752     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6753   }
6754   \CT@arc@
6755   \@@_draw_line:
6756   \endpgfpicture
6757 }
```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```
6758 \cs_new_protected:Npn \@@_hline_v:
6759 {
6760   \begin { tikzpicture }
```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6761 \CT@arc@
6762 \tl_if_empty:NF \l_@@_rule_color_tl
6763 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6764 \pgfrememberpicturepositiononpagetrue
6765 \pgf@relevantforpicturesizefalse
6766 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6767 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6768 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6769 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6770 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6771 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6772 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6773 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6774 ( \l_tmpa_dim , \l_tmpb_dim ) --
6775 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6776 \end { tikzpicture }
6777 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners — if the key `corners` is used).

```

6778 \cs_new_protected:Npn \@@_draw_hlines:
6779 {
6780   \int_step_inline:nnn
6781   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6782   {
6783     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6784     { \c@iRow }
6785     { \int_eval:n { \c@iRow + 1 } }
6786   }
6787   {
6788     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6789     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6790     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6791   }
6792 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6793 \cs_set:Npn \@@_Hline: { \noalign \bgroup \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

6794 \cs_set:Npn \@@_Hline_i:n #1
6795 {
6796   \peek_remove_spaces:n
6797   {
6798     \peek_meaning:NTF \Hline
6799     { \@@_Hline_ii:nn { #1 + 1 } }
6800     { \@@_Hline_iii:n { #1 } }
6801   }
6802 }
6803 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6804 \cs_set:Npn \@@_Hline_iii:n #1
6805 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6806 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6807 {
6808   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6809   \skip_vertical:N \l_@@_rule_width_dim
6810   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

6811 {
6812   \@@_hline:n
6813   {
6814     multiplicity = #1 ,
6815     position = \int_eval:n { \c@iRow + 1 } ,
6816     total-width = \dim_use:N \l_@@_rule_width_dim ,
6817     #2
6818   }
6819 }
6820 \egroup
6821 }

```

### Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of `key=value` pairs.

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

6822 \cs_new_protected:Npn \@@_custom_line:n #1
6823 {
6824   \str_clear_new:N \l_@@_command_str
6825   \str_clear_new:N \l_@@_ccommand_str
6826   \str_clear_new:N \l_@@_letter_str
6827   \tl_clear_new:N \l_@@_other_keys_tl
6828   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6829   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6830   {
6831     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6832   \str_set:Ne \l_@@_command_str
6833   { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6834 }
6835 \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6836 {
6837   \str_set:Ne \l_@@_ccommand_str
6838   { \str_tail:N \l_@@_ccommand_str }
6839   \str_set:Ne \l_@@_ccommand_str
6840   { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6841 }

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6842 \bool_lazy_all:nTF
6843 {
6844   { \str_if_empty_p:N \l_@@_letter_str }
6845   { \str_if_empty_p:N \l_@@_command_str }
6846   { \str_if_empty_p:N \l_@@_ccommand_str }
6847 }
6848 { \@@_error:n { No~letter~and~no~command } }
6849 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6850 }
6851 \keys_define:nn { nicematrix / custom-line }
6852 {
6853   letter .str_set:N = \l_@@_letter_str ,
6854   letter .value_required:n = true ,
6855   command .str_set:N = \l_@@_command_str ,
6856   command .value_required:n = true ,
6857   ccommand .str_set:N = \l_@@_ccommand_str ,
6858   ccommand .value_required:n = true ,
6859 }

```

```

6860 \cs_new_protected:Npn \@@_custom_line_i:n #1
6861 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6862 \bool_set_false:N \l_@@_tikz_rule_bool
6863 \bool_set_false:N \l_@@_dotted_rule_bool
6864 \bool_set_false:N \l_@@_color_bool

6865 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6866 \bool_if:NT \l_@@_tikz_rule_bool
6867 {
6868   \IfPackageLoadedF { tikz }
6869   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6870   \bool_if:NT \l_@@_color_bool
6871   { \@@_error:n { color-in-custom-line-with-tikz } }
6872 }
6873 \bool_if:NT \l_@@_dotted_rule_bool
6874 {
6875   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6876   { \@@_error:n { key-multiplicity-with-dotted } }
6877 }
6878 \str_if_empty:NF \l_@@_letter_str
6879 {
6880   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6881   { \@@_error:n { Several-letters } }
6882   {
6883     \tl_if_in:NoTF
6884     \c_@@_forbidden_letters_str
6885     \l_@@_letter_str
6886     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6887   }

```

During the analysis of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6888 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6889 { \@@_v_custom_line:nn { #1 } }
6890 }
6891 }
6892 }
6893 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6894 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6895 }
6896 \cs_generate_variant:Nn \@@_custom_line_i:n { o }

6897 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6898 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6899 \keys_define:nn { nicematrix / custom-line-bis }
6900 {
6901   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6902   multiplicity .initial:n = 1 ,
6903   multiplicity .value_required:n = true ,
6904   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6905   color .value_required:n = true ,
6906   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6907   tikz .value_required:n = true ,
6908   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6909   dotted .value_forbidden:n = true ,

```

```

6910 total-width .code:n = { } ,
6911 total-width .value_required:n = true ,
6912 width .code:n = { } ,
6913 width .value_required:n = true ,
6914 sep-color .code:n = { } ,
6915 sep-color .value_required:n = true ,
6916 unknown .code:n =
6917   \@@_unknown_key:nn
6918   { nicematrix / custom-line-bis }
6919   { Unknown-key-for-custom-line }
6920 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6921 \bool_new:N \l_@@_dotted_rule_bool
6922 \bool_new:N \l_@@_tikz_rule_bool
6923 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

6924 \keys_define:nn { nicematrix / custom-line-width }
6925 {
6926   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6927   multiplicity .initial:n = 1 ,
6928   multiplicity .value_required:n = true ,
6929   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6930   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6931   \bool_set_true:N \l_@@_total_width_bool ,
6932   total-width .value_required:n = true ,
6933   width .meta:n = { total-width = #1 } ,
6934   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6935 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6936 \cs_new_protected:Npn \@@_h_custom_line:n #1
6937 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign` (which is in `\Hline`).

```

6938   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6939   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6940 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6941 \cs_new_protected:Npn \@@_c_custom_line:n #1
6942 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6943   \exp_args:Nc \NewExpandableDocumentCommand
6944   { nicematrix - \l_@@_ccommand_str }
6945   { 0 { } m }
6946   {
6947     \noalign
6948     {
6949       \@@_compute_rule_width:n { #1 , ##1 }
6950       \skip_vertical:n { \l_@@_rule_width_dim }

```

```

6951         \clist_map_inline:nn
6952         { ##2 }
6953         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6954     }
6955 }
6956 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6957 }

```

The first argument is the list of key-value pairs characteristic of the line. The second argument is the specification of columns for the `\cline` with the syntax *a-b*.

```

6958 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6959 {
6960     \tl_if_in:nnTF { #2 } { - }
6961     { \@@_cut_on_hyphen:w #2 \q_stop }
6962     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6963     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6964     {
6965         \@@_hline:n
6966         {
6967             #1 ,
6968             start = \l_tmpa_tl ,
6969             end = \l_tmpb_tl ,
6970             position = \int_eval:n { \c@iRow + 1 } ,
6971             total-width = \dim_use:N \l_@@_rule_width_dim
6972         }
6973     }
6974 }
6975 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6976 {
6977     \bool_set_false:N \l_@@_tikz_rule_bool
6978     \bool_set_false:N \l_@@_total_width_bool
6979     \bool_set_false:N \l_@@_dotted_rule_bool
6980     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6981     \bool_if:NF \l_@@_total_width_bool
6982     {
6983         \bool_if:NTF \l_@@_dotted_rule_bool
6984         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6985         {
6986             \bool_if:NF \l_@@_tikz_rule_bool
6987             {
6988                 \dim_set:Nn \l_@@_rule_width_dim
6989                 {
6990                     \arrayrulewidth * \l_@@_multiplicity_int
6991                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6992                 }
6993             }
6994         }
6995     }
6996 }

```

The following constructions aims to allow cumulative blocks of options between square brackets such as `I[color=blue][tikz=dashed]`.

```

6997 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6998 {
6999     \str_if_eq:nnTF { #2 } { [ ]
7000     { \@@_v_custom_line_i:nw { #1 } [ ] }
7001     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7002 }
7003 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7004 { \@@_v_custom_line:nn { #1 , #2 } }
7005 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7006 {
7007     \@@_compute_rule_width:n { #2 }

```



In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7008 \tl_gput_right:Ne \g_@@_array_preamble_tl
7009 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
7010 \tl_gput_right:Ne \g_@@_pre_code_after_tl
7011 {
7012   \@@_vline:n
7013   {
7014     #2 ,
7015     position = \int_eval:n { \c@jCol + 1 } ,
7016     total-width = \dim_use:N \l_@@_rule_width_dim
7017   }
7018 }
7019 \@@_rec_preamble:n #1
7020 }
7021 \@@_custom_line:n
7022 { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

## The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

7023 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7024 {
7025   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
7026   {
7027     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7028     {
7029       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
7030       {
7031         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7032         { \bool_gset_false:N \g_tmpa_bool }
7033       }
7034     }
7035   }
7036 }

```

The same for vertical rules.

```

7037 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7038 {
7039   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
7040   {
7041     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7042     {
7043       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
7044       {
7045         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7046         { \bool_gset_false:N \g_tmpa_bool }
7047       }
7048     }
7049   }
7050 }
7051 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7052 {
7053   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
7054   {
7055     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7056     {
7057       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
7058       { \bool_gset_false:N \g_tmpa_bool }
7059     }

```

```

7060         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
7061         { \bool_gset_false:N \g_tmpa_bool }
7062     }
7063 }
7064 }
7065 }
7066 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7067 {
7068     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
7069     {
7070         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7071         {
7072             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
7073             { \bool_gset_false:N \g_tmpa_bool }
7074             {
7075                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
7076                 { \bool_gset_false:N \g_tmpa_bool }
7077             }
7078         }
7079     }
7080 }

```

## 23 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

7081 \cs_new_protected:Npn \@@_compute_corners:
7082 {
7083     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7084     { \@@_mark_cells_of_block:nnnnn ##1 }

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

7085     \clist_clear:N \l_@@_corners_cells_clist
7086     \clist_map_inline:Nn \l_@@_corners_clist
7087     {
7088         \str_case:nnF { ##1 }
7089         {
7090             { NW }
7091             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7092             { NE }
7093             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7094             { SW }
7095             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7096             { SE }
7097             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7098         }
7099         { \@@_error:nn { bad~corner } { ##1 } }
7100     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7101     \clist_if_empty:NF \l_@@_corners_cells_clist
7102     {

```

You write on the aux file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which colors the `rows`, `columns` and `cells` must not color the cells in the corners.

```

7103     \tl_gput_right:Ne \g_@@_aux_tl
7104     {
7105         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7106         { \l_@@_corners_cells_clist }
7107     }
7108 }
7109 }

7110 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7111 {
7112     \int_step_inline:nnn { #1 } { #3 }
7113     {
7114         \int_step_inline:nnn { #2 } { #4 }
7115         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7116     }
7117 }

7118 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7119 {
7120     \cs_if_exist:cTF
7121     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7122     { \prg_return_true: }
7123     { \prg_return_false: }
7124 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_clist`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```

7125 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7126 {

```

For the explanations and the name of the variables, we consider that we are computing the left-upper corner.

First, we try to determine which is the last empty cell (and not in a block: we won’t add that precision any longer) in the column of number 1. The flag `\l_tmpa_bool` will be raised when a non-empty cell is found.

```

7127     \bool_set_false:N \l_tmpa_bool
7128     \int_zero_new:N \l_@@_last_empty_row_int
7129     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7130     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7131     {
7132         \bool_lazy_or:nnTF
7133         {
7134             \cs_if_exist_p:c
7135             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7136         }
7137         { \@@_if_in_block_p:nn { ##1 } { #2 } }
7138         { \bool_set_true:N \l_tmpa_bool }
7139     }
7140     \bool_if:NF \l_tmpa_bool
7141     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7142 }
7143 }

```

Now, you determine the last empty cell in the row of number 1.

```

7144 \bool_set_false:N \l_tmpa_bool
7145 \int_zero_new:N \l_@@_last_empty_column_int
7146 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7147 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7148 {
7149   \bool_lazy_or:nnTF
7150   {
7151     \cs_if_exist_p:c
7152     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7153   }
7154   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7155   { \bool_set_true:N \l_tmpa_bool }
7156   {
7157     \bool_if:NF \l_tmpa_bool
7158     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7159   }
7160 }

```

Now, we loop over the rows.

```

7161 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
7162 {

```

We treat the row number ##1 with another loop.

```

7163 \bool_set_false:N \l_tmpa_bool
7164 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7165 {
7166   \bool_lazy_or:nnTF
7167   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7168   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7169   { \bool_set_true:N \l_tmpa_bool }
7170   {
7171     \bool_if:NF \l_tmpa_bool
7172     {
7173       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7174       \clist_put_right:Nn
7175       \l_@@_corners_cells_clist
7176       { ##1 - #####1 }
7177       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7178     }
7179   }
7180 }
7181 }
7182 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7183 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7184 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...`

## 24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7185 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

7186 \keys_define:nn { nicematrix / NiceMatrixBlock }
7187 {
7188   auto-columns-width .code:n =
7189   {
7190     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7191     \dim_gzero_new:N \g_@@_max_cell_width_dim
7192     \bool_set_true:N \l_@@_auto_columns_width_bool
7193   }
7194 }

7195 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7196 {
7197   \int_gincr:N \g_@@_NiceMatrixBlock_int
7198   \dim_zero:N \l_@@_columns_width_dim
7199   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7200   \bool_if:NT \l_@@_block_auto_columns_width_bool
7201   {
7202     \cs_if_exist:cT
7203     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7204     {
7205       \dim_set:Nn \l_@@_columns_width_dim
7206       {
7207         \use:c
7208         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7209       }
7210     }
7211   }
7212 }

```

At the end of the environment `{NiceMatrixBlock}`, we write in the main `aux` file instructions for the column width of all the environments of the block (that's why we have stored the number of the first environment of the block in the counter `\l_@@_first_env_block_int`).

```

7213 {
7214   \legacy_if:nTF { measuring@ }

```

If `{NiceMatrixBlock}` is used in an environment of `amsmath` such as `{align}`: cf. question 694957 on TeX StackExchange. The most important line in that case is the following one.

```

7215   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7216   {
7217     \bool_if:NT \l_@@_block_auto_columns_width_bool
7218     {
7219       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7220       \iow_shipout:Ne \@mainaux
7221       {
7222         \cs_gset:cpn
7223         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

7224         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7225       }
7226       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7227     }
7228   }
7229   \ignorespacesafterend
7230 }

```

## 25 The extra nodes

The following command is called in `\@@_use_arraybox_with_notes_c`: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

7231 \cs_new_protected:Npn \@@_create_extra_nodes:
7232 {
7233   \bool_if:nTF \l_@@_medium_nodes_bool
7234   {
7235     \bool_if:NTF \l_@@_no_cell_nodes_bool
7236     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7237     {
7238       \bool_if:NTF \l_@@_large_nodes_bool
7239       \@@_create_medium_and_large_nodes:
7240       \@@_create_medium_nodes:
7241     }
7242   }
7243   {
7244     \bool_if:NT \l_@@_large_nodes_bool
7245     {
7246       \bool_if:NTF \l_@@_no_cell_nodes_bool
7247       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7248       \@@_create_large_nodes:
7249     }
7250   }
7251 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row  $i$ , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal  $y$ -value of all the cells of the row  $i$ . The dimension `l_@@_row_i_max_dim` is the maximal  $y$ -value of all the cells of the row  $i$ .

Similarly, for each column  $j$ , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal  $x$ -value of all the cells of the column  $j$ . The dimension `l_@@_column_j_max_dim` is the maximal  $x$ -value of all the cells of the column  $j$ .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

7252 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7253 {
7254   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7255   {
7256     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7257     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7258     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7259     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7260   }
7261   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7262   {
7263     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7264     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7265     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7266     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7267   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7268 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7269 {
7270     \int_step_variable:nnNn
7271     \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

If the cell ( $i$ - $j$ ) is empty or an implicit cell (that is to say a cell after implicit ampersands &) we don't update the dimensions we want to compute.

```

7272 {
7273     \cs_if_exist:cT
7274     { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell ( $i$ - $j$ ). They will be stored in `\pgf@x` and `\pgf@y`.

```

7275 {
7276     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7277     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7278     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7279     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7280     {
7281         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7282         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7283     }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell ( $i$ - $j$ ). They will be stored in `\pgf@x` and `\pgf@y`.

```

7284     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7285     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7286     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7287     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7288     {
7289         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7290         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7291     }
7292 }
7293 }
7294 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7295 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7296 {
7297     \dim_compare:nNnT
7298     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7299     {
7300         \@@_qpoint:n { row - \@@_i: - base }
7301         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7302         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7303     }
7304 }
7305 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7306 {
7307     \dim_compare:nNnT
7308     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7309     {
7310         \@@_qpoint:n { col - \@@_j: }
7311         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7312         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7313     }
7314 }
7315 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7316 \cs_new_protected:Npn \@@_create_medium_nodes:
7317 {
7318   \pgfpicture
7319   \pgfrememberpicturepositiononpagetrue
7320   \pgf@relevantforpicturesizefalse
7321   \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

7322   \tl_set:Nn \l_@@_suffix_tl { -medium }
7323   \@@_create_nodes:
7324   \endpgfpicture
7325 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>15</sup>. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7326 \cs_new_protected:Npn \@@_create_large_nodes:
7327 {
7328   \pgfpicture
7329   \pgfrememberpicturepositiononpagetrue
7330   \pgf@relevantforpicturesizefalse
7331   \@@_computations_for_medium_nodes:
7332   \@@_computations_for_large_nodes:
7333   \tl_set:Nn \l_@@_suffix_tl { - large }
7334   \@@_create_nodes:
7335   \endpgfpicture
7336 }
7337 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7338 {
7339   \pgfpicture
7340   \pgfrememberpicturepositiononpagetrue
7341   \pgf@relevantforpicturesizefalse
7342   \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

7343   \tl_set:Nn \l_@@_suffix_tl { - medium }
7344   \@@_create_nodes:
7345   \@@_computations_for_large_nodes:
7346   \tl_set:Nn \l_@@_suffix_tl { - large }
7347   \@@_create_nodes:
7348   \endpgfpicture
7349 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

7350 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7351 {
7352   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7353   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

7354   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7355   {
7356     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`



```

7357     {
7358     (
7359         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7360         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7361     )
7362     / 2
7363     }
7364     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7365     { l_@@_row _ \@@_i: _ min_dim }
7366 }
7367 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7368 {
7369     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7370     {
7371     (
7372         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7373         \dim_use:c
7374         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7375     )
7376     / 2
7377     }
7378     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7379     { l_@@_column _ \@@_j: _ max _ dim }
7380 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7381     \dim_sub:cn
7382     { l_@@_column _ 1 _ min _ dim }
7383     \l_@@_left_margin_dim
7384     \dim_add:cn
7385     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7386     \l_@@_right_margin_dim
7387 }

```

The command `\@@_create_nodes:` is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses `\l_@@_suffix_tl` (-medium or -large).

```

7388 \cs_new_protected:Npn \@@_create_nodes:
7389 {
7390     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7391     {
7392         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7393         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7394         \@@_pgf_rect_node:nnnnn
7395         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7396         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7397         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7398         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7399         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7400         \str_if_empty:NF \l_@@_name_str
7401         {
7402             \pgfnodealias
7403             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7404             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7405         }
7406     }
7407 }
7408 \int_step_inline:nn { \c@iRow }

```

```

7409 {
7410   \pgfnodealias
7411   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7412   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7413 }
7414 \int_step_inline:nn { \c@jCol }
7415 {
7416   \pgfnodealias
7417   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7418   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7419 }
7420 \pgfnodealias % added 2025-04-05
7421 { \@@_env: - last - last \l_@@_suffix_tl }
7422 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol \l_@@_suffix_tl }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with  $n > 1$  was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of  $n$ .

```

7423   \seq_map_pairwise_function:NNN
7424   \g_@@_multicolumn_cells_seq
7425   \g_@@_multicolumn_sizes_seq
7426   \@@_node_for_multicolumn:nn
7427 }

7428 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7429 {
7430   \cs_set_nopar:Npn \@@_i: { #1 }
7431   \cs_set_nopar:Npn \@@_j: { #2 }
7432 }

```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format  $i$ - $j$  and the second is the value of  $n$  (the length of the “multi-cell”).

```

7433 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7434 {
7435   \@@_extract_coords_values: #1 \q_stop
7436   \@@_pgf_rect_node:nnnnn
7437   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7438   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7439   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7440   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7441   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7442   \str_if_empty:NF \l_@@_name_str
7443   {
7444     \pgfnodealias
7445     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7446     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7447   }
7448 }

```

## 26 The blocks

The following code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass (in the cell of the array).

```

7449 \keys_define:nn { nicematrix / Block / FirstPass }
7450 {
7451   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7452     \bool_set_true:N \l_@@_p_block_bool ,
7453   j .value_forbidden:n = true ,
7454   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7455   l .value_forbidden:n = true ,
7456   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7457   r .value_forbidden:n = true ,
7458   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7459   c .value_forbidden:n = true ,
7460   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7461   L .value_forbidden:n = true ,
7462   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7463   R .value_forbidden:n = true ,
7464   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7465   C .value_forbidden:n = true ,
7466   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7467   t .value_forbidden:n = true ,
7468   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7469   T .value_forbidden:n = true ,
7470   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7471   b .value_forbidden:n = true ,
7472   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7473   B .value_forbidden:n = true ,
7474   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7475   m .value_forbidden:n = true ,
7476   v-center .meta:n = m ,
7477   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7478   p .value_forbidden:n = true ,
7479   color .code:n =
7480     \@@_color:n { #1 }
7481     \tl_set_rescan:Nnn
7482       \l_@@_draw_tl
7483       { \char_set_catcode_other:N ! }
7484       { #1 } ,
7485   color .value_required:n = true ,
7486   respect-arraystretch .code:n =
7487     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7488   respect-arraystretch .value_forbidden:n = true ,
7489 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>`. It's mandatory to use an expandable command.

```

7490 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7491 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7492 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax *i-j*) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

7493   \tl_if_blank:nTF { #2 }
7494   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7495   {
7496     \tl_if_in:nnTF { #2 } { - }
7497     {
7498       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7499       \@@_Block_i_czech:w \@@_Block_i:w
7500       #2 \q_stop
7501     }
7502     {
7503       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7504         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7505     }
7506 }
7507 { #1 } { #3 } { #4 }
7508 \ignorespaces
7509 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7510 \cs_new:Npn \@@_Block_i:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

With `babel` with the key `czech`, the character `-` (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command `\@@_Block:` to do the job because the command `\@@_Block:` is defined with the command `\NewExpandableDocumentCommand`.

```

7511 {
7512   \char_set_catcode_active:N -
7513   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7514 }

```

Now, the arguments have been extracted: `#1` is  $i$  (the number of rows of the block), `#2` is  $j$  (the number of columns of the block), `#3` is the list of `key=values` pairs, `#4` are the tokens to put before the math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7515 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7516 {

```

We recall that `#1` and `#2` have been extracted from the first mandatory argument of `\Block` (which is of the syntax  $i-j$ ). However, the user is allowed to omit  $i$  or  $j$  (or both). We detect that situation by replacing a missing value by 100 (it's a convention: when the block will actually be drawn these values will be detected and interpreted as *maximal possible value* according to the actual size of the array).

```

7517   \bool_lazy_or:nnTF
7518     { \tl_if_blank_p:n { #1 } }
7519     { \str_if_eq_p:ee { * } { #1 } }
7520     { \int_set:Nn \l_tmpa_int { 100 } }
7521     { \int_set:Nn \l_tmpa_int { #1 } }
7522   \bool_lazy_or:nnTF
7523     { \tl_if_blank_p:n { #2 } }
7524     { \str_if_eq_p:ee { * } { #2 } }
7525     { \int_set:Nn \l_tmpb_int { 100 } }
7526     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7527   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7528   {
7529     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7530     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7531     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7532   }
7533   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

7534   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7535   \tl_set:Ne \l_tmpa_tl
7536   {
7537     { \int_use:N \c@iRow }
7538     { \int_use:N \c@jCol }
7539     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7540     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7541   }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:  
`{imin}{jmin}{imax}{jmax}`.

We have different treatments when the key `p` is used and when the block is mono-column or mono-row, etc. That’s why we have several macros: `\@@_Block_iv:nnnnn`, `\@@_Block_v:nnnnn`, `\@@_Block_vi:nnnn`, etc. (the five arguments of those macros are provided by curryfication).

```

7542     \bool_set_false:N \l_tmpa_bool
7543     \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7544     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7545     \bool_case:nF
7546     {
7547         \l_tmpa_bool                                { \@@_Block_vii:eennn }
7548         \l_@@_p_block_bool                          { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7549         \l_@@_X_bool                                { \@@_Block_v:eennn }
7550         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7551         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7552         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7553     }
7554     { \@@_Block_v:eennn }
7555     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7556 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

**#1** is *i* (the number of rows of the block), **#2** is *j* (the number of columns of the block), **#3** is the list of `key=values` pairs, **#4** are the tokens to put before the potential math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

7557 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7558 {
7559     \int_gincr:N \g_@@_block_box_int
7560     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7561     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7562     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7563     {
7564         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7565         {
7566             \@@_actually_diagbox:nnnnnn
7567             { \int_use:N \c@iRow }
7568             { \int_use:N \c@jCol }
7569             { \int_eval:n { \c@iRow + #1 - 1 } }
7570             { \int_eval:n { \c@jCol + #2 - 1 } }
7571             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7572             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7573         }
7574     }
7575     \box_gclear_new:c
7576     { g_@@_block_box_int _int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7577 \hbox_gset:cn
7578 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7579 {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current`: (in order to use `\color_ensure_current`: safely, you should load `l3backend` before the `\documentclass`).

```

7580 \tl_if_empty:NTF \l_@@_color_tl
7581 { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7582 { \@@_color:o \l_@@_color_tl }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

7583 \int_compare:nNnT { #1 } = { \c_one_int }
7584 {
7585   \int_if_zero:nTF { \c@iRow }
7586   {

```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & & -4 & 5 & \\
3 & & -4 & 5 & & -6 & \\
-4 & 5 & & -6 & 7 & & \\
5 & & -6 & 7 & & -8 & \\
\end{bNiceMatrix}$

```

```

7587 \cs_set_eq:NN \Block \@@_NullBlock:
7588 \l_@@_code_for_first_row_tl
7589 }
7590 {
7591   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7592   {
7593     \cs_set_eq:NN \Block \@@_NullBlock:
7594     \l_@@_code_for_last_row_tl
7595   }
7596 }
7597 \g_@@_row_style_tl
7598 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7599 \@@_reset_arraystretch:
7600 \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7601 #4

```

We adjust `\l_@@_hpos_block_str` when `\rotate` has been used (in the cell where the command `\Block` is used but maybe in #4, `\RowStyle`, `code-for-first-row`, etc.).

```
7602 \@@_adjust_hpos_rotate:
```

The boolean `\g_@@_rotate_bool` will be also considered *after the composition of the box* (in order to rotate the box).

Remind that we are in the command of composition of the box of the block. Previously, we have only done some tuning. Now, we will actually compose the content with a `{tabular}`, an `{array}` or a `{minipage}`.

```
7603 \bool_if:NTF \l_@@_tabular_bool
7604 {
7605   \bool_lazy_all:nTF
7606   {
7607     { \int_compare_p:nNn { #2 } = { \c_one_int } }
```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of  $-1$  cm.

```
7608   {
7609     ! \dim_compare_p:nNn
7610     { \l_@@_col_width_dim } < { \c_zero_dim }
7611   }
7612   { ! \g_@@_rotate_bool }
7613 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7614 {
7615   \use:e
7616   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7617 \exp_not:N \begin { minipage }
7618 [ \str_lowercase:f \l_@@_vpos_block_str ]
7619 { \l_@@_col_width_dim }
7620 \str_case:on \l_@@_hpos_block_str
7621 { c \centering r \raggedleft l \raggedright }
7622 }
7623 #5
7624 \end { minipage }
7625 }
```

In the other cases, we use a `{tabular}`.

```
7626 {
7627   \use:e
7628   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7629 \exp_not:N \begin { tabular }
7630 [ \str_lowercase:f \l_@@_vpos_block_str ]
7631 { @ { } \l_@@_hpos_block_str @ { } }
7632 }
7633 #5
7634 \end { tabular }
7635 }
7636 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7637 {
7638   $ % $
7639   \use:e
7640   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7641         \exp_not:N \begin { array }
7642         [ \str_lowercase:f \l_@@_vpos_block_str ]
7643         { @ { } \l_@@_hpos_block_str @ { } }
7644     }
7645     #5
7646     \end { array }
7647     $ % $
7648 }
7649 }
```

The box which will contain the content of the block has now been composed.

If there were `\rotate` (which raises `\g_@@_rotate_bool`) in the content of the `\Block`, we do a rotation of the box (and we also adjust the baseline of the rotated box).

```

7650     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_box_of_block: }
```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

7651     \int_compare:nNnT { #2 } = { \c_one_int }
7652     {
7653         \dim_gset:Nn \g_@@_blocks_wd_dim
7654         {
7655             \dim_max:nn
7656             { \g_@@_blocks_wd_dim }
7657             {
7658                 \box_wd:c
7659                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7660             }
7661         }
7662     }
```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7663     \int_compare:nNnT { #1 } = { \c_one_int }
7664     {
7665         \bool_lazy_any:nT
7666         {
7667             { \str_if_empty_p:N \l_@@_vpos_block_str }
7668             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7669             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7670         }
7671         { \@@_adjust_blocks_ht_dp: }
7672     }
7673     \seq_gput_right:Ne \g_@@_blocks_seq
7674     {
7675         \l_tmpa_tl
```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in `\l_@@_hpos_block_str`. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of `\l_@@_hpos_block_str`, which is fixed by the type of current column.

```

7676     {
7677         \exp_not:n { #3 } ,
7678         \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

7679     \bool_if:NT \g_@@_rotate_bool
7680     {
7681         \bool_if:NTF \g_@@_rotate_c_bool
7682         { m }
7683         {
```



```

7684         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7685         { T }
7686     }
7687 }
7688 }
7689 {
7690     \box_use_drop:c
7691     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7692 }
7693 }
7694 \bool_set_false:N \g_@@_rotate_c_bool
7695 }

7696 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7697 {
7698     \dim_gset:Nn \g_@@_blocks_ht_dim
7699     {
7700         \dim_max:nn
7701         { \g_@@_blocks_ht_dim }
7702         {
7703             \box_ht:c
7704             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7705         }
7706     }
7707     \dim_gset:Nn \g_@@_blocks_dp_dim
7708     {
7709         \dim_max:nn
7710         { \g_@@_blocks_dp_dim }
7711         {
7712             \box_dp:c
7713             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7714         }
7715     }
7716 }

7717 \cs_new:Npn \@@_adjust_hpos_rotate:
7718 {
7719     \bool_if:NT \g_@@_rotate_bool
7720     {
7721         \str_set:Ne \l_@@_hpos_block_str
7722         {
7723             \bool_if:NTF \g_@@_rotate_c_bool
7724             { c }
7725             {
7726                 \str_case:onF \l_@@_vpos_block_str
7727                 { b l B l t r T r }
7728                 {
7729                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7730                     { r }
7731                     { l }
7732                 }
7733             }
7734         }
7735     }
7736 }
7737 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7738 \cs_new_protected:Npn \@@_rotate_box_of_block:
7739 {
7740     \box_grotate:cn

```

```

7741 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7742 { 90 }
7743 \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
7744 {
7745   \vbox_gset_top:cn
7746   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7747   {
7748     \skip_vertical:n { 0.8 ex }
7749     \box_use:c
7750     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7751   }
7752 }
7753 \bool_if:NT \g_@@_rotate_c_bool
7754 {
7755   \hbox_gset:cn
7756   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7757   {
7758     $ % $
7759     \vcenter
7760     {
7761       \box_use:c
7762       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7763     }
7764     $ % $
7765   }
7766 }
7767 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is  $i$  (the number of rows of the block), #2 is  $j$  (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7768 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7769 {
7770   \seq_gput_right:Ne \g_@@_blocks_seq
7771   {
7772     \l_tmpa_tl
7773     { \exp_not:n { #3 } }
7774     {
7775       \bool_if:NTF \l_@@_tabular_bool
7776       {
7777         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7778 \@@_reset_arraystretch:
7779 \exp_not:n
7780 {
7781   \dim_zero:N \extrarowheight
7782   #4

```

If the box is rotated (the key `\rotate` may be in the previous #4), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

7783 \tag_if_active:T { \tag_stop:n { table } }
7784 \use:e
7785 {
7786   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7787   { @ { } \l_@@_hpos_block_str @ { } }

```

```

7788         }
7789         #5
7790     \end { tabular }
7791 }
7792 \group_end:
7793 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7794 {
7795     \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7796     \@@_reset_arraystretch:
7797     \exp_not:n
7798     {
7799         \dim_zero:N \extrarowheight
7800         #4
7801         $ % $
7802         \use:e
7803         {
7804             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7805             { @ { } \l_@@_hpos_block_str @ { } }
7806         }
7807         #5
7808         \end { array }
7809         $ % $
7810     }
7811     \group_end:
7812 }
7813 }
7814 }
7815 }
7816 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

7817 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7818 {
7819     \seq_gput_right:Ne \g_@@_blocks_seq
7820     {
7821         \l_tmpa_tl
7822         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7823         { { \exp_not:n { #4 #5 } } }
7824     }
7825 }
7826 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }

```

The following macro is also for the case of a \Block which uses the key p.

```

7827 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7828 {
7829     \seq_gput_right:Ne \g_@@_blocks_seq
7830     {
7831         \l_tmpa_tl
7832         { \exp_not:n { #3 } }
7833         { \exp_not:n { #4 #5 } }
7834     }
7835 }
7836 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }

```

We recall that the options of the command \Block are analyzed twice: first in the cell of the array and once again when the block will be put in the array *after the construction of the array* (by using PGF).

```

7837 \keys_define:n { nicematrix / Block / SecondPass }
7838 {
7839     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7840     ampersand-in-blocks .default:n = true ,
7841     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7842     tikz .code:n =
7843         \IfPackageLoadedTF { tikz }
7844         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7845         { \@@_error:n { tikz~key~without~tikz } } ,
7846     tikz .value_required:n = true ,
7847     fill .code:n =
7848         \tl_set_rescan:Nnn
7849         \l_@@_fill_tl
7850         { \char_set_catcode_other:N ! }
7851         { #1 } ,
7852     fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

7853     opacity .tl_set:N = \l_@@_opacity_tl ,
7854     opacity .value_required:n = true ,
7855     draw .code:n =
7856         \tl_set_rescan:Nnn
7857         \l_@@_draw_tl
7858         { \char_set_catcode_other:N ! }
7859         { #1 } ,
7860     draw .default:n = default ,
7861     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7862     rounded-corners .default:n = 4 pt ,
7863     color .code:n =
7864         \@@_color:n { #1 }
7865         \tl_set_rescan:Nnn
7866         \l_@@_draw_tl
7867         { \char_set_catcode_other:N ! }
7868         { #1 } ,
7869     borders .clist_set:N = \l_@@_borders_clist ,
7870     borders .value_required:n = true ,
7871     hvlines .meta:n = { vlines , hlines } ,
7872     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7873     vlines .default:n = true ,
7874     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7875     hlines .default:n = true ,
7876     line-width .dim_set:N = \l_@@_line_width_dim ,
7877     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7878     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7879         \bool_set_true:N \l_@@_p_block_bool ,
7880     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7881     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7882     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7883     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7884         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7885     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7886         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7887     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7888         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7889     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7890     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7891     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7892     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7893     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7894     m .value_forbidden:n = true ,
7895     v-center .meta:n = m ,

```

```

7896 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7897 p .value_forbidden:n = true ,
7898 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7899 name .value_required:n = true ,
7900 name .initial:n = ,
7901 respect-arraystretch .code:n =
7902   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7903 respect-arraystretch .value_forbidden:n = true ,
7904 transparent .bool_set:N = \l_@@_transparent_bool ,
7905 transparent .default:n = true ,
7906 transparent .initial:n = false ,
7907 unknown .code:n =
7908   \@@_unknown_key:nn
7909     { nicematrix / Block / SecondPass }
7910     { Unknown-key-for-Block }
7911 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7912 \cs_new_protected:Npn \@@_draw_blocks:
7913 {
7914   \bool_if:NTF \c_@@_revtex_bool
7915     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7916     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7917   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7918 }
7919 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7920 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7921   \int_zero:N \l_@@_last_row_int
7922   \int_zero:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format  $i-j$ . However, the user is allowed to omit  $i$  or  $j$  (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7923   \int_compare:nNnTF { #3 } > { 98 }
7924     { \int_set_eq:NN \l_@@_last_row_int \c_iRow }
7925     { \int_set:Nn \l_@@_last_row_int { #3 } }
7926   \int_compare:nNnTF { #4 } > { 98 }
7927     { \int_set_eq:NN \l_@@_last_col_int \c_jCol }
7928     { \int_set:Nn \l_@@_last_col_int { #4 } }
7929   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7930     {
7931       \bool_lazy_and:nnTF
7932         { \l_@@_preamble_bool }
7933         {
7934           \int_compare_p:n
7935             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7936         }
7937       {
7938         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7939         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7940         \@@_msg_redirect_name:nn { columns-not-used } { none }
7941       }
7942     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7943   }

```

```

7944 {
7945   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7946   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7947   {
7948     \@@_Block_v:nneenn
7949     { #1 }
7950     { #2 }
7951     { \int_use:N \l_@@_last_row_int }
7952     { \int_use:N \l_@@_last_col_int }
7953     { #5 }
7954     { #6 }
7955   }
7956 }
7957 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of `key=value` options; #6 is the label (content) of the block.

```

7958 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7959 {

```

The group is for the keys.

```

7960   \group_begin:
7961   \int_compare:nNnT { #1 } = { #3 }
7962   { \str_set:Nn \l_@@_vpos_block_str { t } }
7963   \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7964   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7965   \bool_lazy_and:nnT
7966   { \l_@@_vlines_block_bool }
7967   { ! \l_@@_ampersand_bool }
7968   {
7969     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7970     {
7971       \@@_vlines_block:nnn
7972       { \exp_not:n { #5 } }
7973       { #1 - #2 }
7974       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7975     }
7976   }
7977   \bool_if:NT \l_@@_hlines_block_bool
7978   {
7979     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7980     {
7981       \@@_hlines_block:nnn
7982       { \exp_not:n { #5 } }
7983       { #1 - #2 }
7984       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7985     }
7986   }
7987   \bool_if:NF \l_@@_transparent_bool
7988   {
7989     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7990     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7991     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7992     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7993   }
7994 }

```

```

7995 \tl_if_empty:NF \l_@@_draw_tl
7996 {
7997   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7998   { \@@_error:n { hlines~with~color } }
7999   \tl_gput_right:Ne \g_nicematrix_code_after_tl
8000   {
8001     \@@_stroke_block:nnn
8002     { \exp_not:n { #5 } }
8003     { #1 - #2 }
8004     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
8005   }
8006   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8007   { { #1 } { #2 } { #3 } { #4 } }
8008 }
8009 \clist_if_empty:NF \l_@@_borders_clist
8010 {
8011   \tl_gput_right:Ne \g_nicematrix_code_after_tl
8012   {
8013     \@@_stroke_borders_block:nnn
8014     { \exp_not:n { #5 } }
8015     { #1 - #2 }
8016     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
8017   }
8018 }
8019 \tl_if_empty:NF \l_@@_fill_tl
8020 {
8021   \@@_add_opacity_to_fill:
8022   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8023   {
8024     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8025     { #1 - #2 }
8026     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
8027     { \dim_use:N \l_@@_rounded_corners_dim }
8028   }
8029 }
8030 \seq_if_empty:NF \l_@@_tikz_seq
8031 {
8032   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8033   {
8034     \@@_block_tikz:nnnnn
8035     { \seq_use:Nn \l_@@_tikz_seq { , } }
8036     { #1 }
8037     { #2 }
8038     { \int_use:N \l_@@_last_row_int }
8039     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of TikZ keys.

```

8040   }
8041 }
8042 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8043 {
8044   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8045   {
8046     \@@_actually_diagbox:nnnnnn
8047     { #1 }
8048     { #2 }
8049     { \int_use:N \l_@@_last_row_int }
8050     { \int_use:N \l_@@_last_col_int }
8051     { \exp_not:n { ##1 } }
8052     { \exp_not:n { ##2 } }

```

```

8053     }
8054 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one    & \\
                        &      & two    & \\
three                  & four & five    & \\
six                    & seven & eight   & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

|           |       |       |
|-----------|-------|-------|
| our block |       | one   |
|           |       | two   |
| three     | four  | five  |
| six       | seven | eight |

We highlight the node `1-1-block-short`

|           |       |       |
|-----------|-------|-------|
| our block |       | one   |
|           |       | two   |
| three     | four  | five  |
| six       | seven | eight |

The construction of the node corresponding to the merged cells.

```

8055 \pgfpicture
8056 \pgfrememberpicturepositiononpagetrue
8057 \pgf@relevantforpicturesizefalse
8058 \@@_qpoint:n { row - #1 }
8059 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8060 \@@_qpoint:n { col - #2 }
8061 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8062 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8063 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8064 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8065 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

We construct the node for the block with the name `(#1-#2-block)`.

The function `\@@_pgf_rect_node:nnnnn` takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

8066 \@@_pgf_rect_node:nnnnn
8067 { \@@_env: - #1 - #2 - block }
8068 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8069 \str_if_empty:NF \l_@@_block_name_str
8070 {
8071   \pgfnodealias
8072   { \@@_env: - \l_@@_block_name_str }
8073   { \@@_env: - #1 - #2 - block }
8074   \str_if_empty:NF \l_@@_name_str
8075   {
8076     \pgfnodealias
8077     { \l_@@_name_str - \l_@@_block_name_str }
8078     { \@@_env: - #1 - #2 - block }
8079   }
8080 }

```

Now, we create the “short node” which, in general, will be used to put the label (that is to say the content of the node). However, if one the keys `L`, `C` or `R` is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don't need to create that node since the normal node is used to put the label.

```

8081 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8082 {
8083   \dim_set_eq:NN \l_tmpb_dim \c_max_dim

```



The short node is constructed by taking into account the *contents* of the columns involved in at least one cell of the block. That's why we have to do a loop over the rows of the array.

```
8084 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8085 {
```

We recall that, when a cell is empty, no (normal) node is created in that cell. That's why we test the existence of the node before using it.

```
8086 \cs_if_exist:cT
8087 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8088 {
8089 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8090 {
8091 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8092 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8093 }
8094 }
8095 }
```

If all the cells of the column were empty, `\l_tmpb_dim` has still the same value `\c_max_dim`. In that case, you use for `\l_tmpb_dim` the value of the position of the vertical rule.

```
8096 \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
8097 {
8098 \@@_qpoint:n { col - #2 }
8099 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8100 }
8101 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8102 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8103 {
8104 \cs_if_exist:cT
8105 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8106 {
8107 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8108 {
8109 \pgfpointanchor
8110 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8111 { east }
8112 \dim_set:Nn \l_@@_tmpd_dim
8113 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
8114 }
8115 }
8116 }
8117 \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
8118 {
8119 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8120 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8121 }
8122 \@@_pgf_rect_node:nnnn
8123 { \@@_env: - #1 - #2 - block - short }
8124 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8125 }
```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```
8126 \bool_if:NT \l_@@_medium_nodes_bool
8127 {
8128 \@@_pgf_rect_node:nnn
8129 { \@@_env: - #1 - #2 - block - medium }
8130 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8131 {
8132 \pgfpointanchor
8133 { \@@_env:
8134 - \int_use:N \l_@@_last_row_int
8135 - \int_use:N \l_@@_last_col_int - medium
8136 }
```

```

8137         { south-east }
8138     }
8139 }
8140 \endpgfpicture
8141

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8142 \bool_if:NTF \l_@@_ampersand_bool
8143 {
8144     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8145     \int_zero_new:N \l_@@_split_int
8146     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell.

```

8147     \int_zero_new:N \l_@@_first_row_int
8148     \int_zero_new:N \l_@@_first_col_int
8149     \int_zero_new:N \l_@@_last_row_int
8150     \int_zero_new:N \l_@@_last_col_int
8151     \int_set:Nn \l_@@_first_row_int { #1 }
8152     \int_set:Nn \l_@@_first_col_int { #2 }
8153     \int_set:Nn \l_@@_last_row_int { #3 }
8154     \int_set:Nn \l_@@_last_col_int { #4 }

8155     \pgfpicture
8156     \pgfrememberpicturerepositiononpagetrue
8157     \pgf@relevantforpicturesizefalse
8158     \@@_qpoint:n { row - #1 }
8159     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8160     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8161     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8162     \@@_qpoint:n { col - #2 }
8163     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8164     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8165     \dim_set:Nn \l_tmpb_dim
8166     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8167     \bool_lazy_or:nnT
8168     { \l_@@_vlines_block_bool }
8169     { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8170     {
8171         \int_step_inline:nn { \l_@@_split_int - 1 }
8172         {
8173             \pgfpathmoveto
8174             {
8175                 \pgfpoint
8176                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8177                 \l_@@_tmpc_dim
8178             }
8179             \pgfpathlineto
8180             {
8181                 \pgfpoint
8182                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8183                 \l_@@_tmpd_dim
8184             }
8185             \CT@arc@
8186             \pgfsetlinewidth { 1.1 \arrayrulewidth }
8187             \pgfsetrectcap
8188             \pgfusepathqstroke
8189         }
8190     }
8191     \cs_set_eq:NN \cellcolor \@@_subcellcolor
8192     \int_zero_new:N \l_@@_split_i_int
8193     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }

```

```

8194 {
8195   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8196   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8197 }
8198 {
8199   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8200   {
8201     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8202     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8203   }
8204   {
8205     \bool_lazy_or:nnTF
8206     { \int_compare_p:nNn { #1 } = { #3 } }
8207     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8208     {
8209       \@@_qpoint:n { row - #1 - base }
8210       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8211     }
8212     {
8213       \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8214       {
8215         \@@_qpoint:n { row - #3 - base }
8216         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8217       }
8218       {
8219         \@@_qpoint:n { #1 - #2 - block }
8220         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8221       }
8222     }
8223   }
8224 }
8225 \int_step_inline:nn { \l_@@_split_int }
8226 {
8227   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8228   \int_set:Nn \l_@@_split_i_int { ##1 }
8229   \dim_set:Nn \col@sep
8230   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8231   \pgftransformshift
8232   {
8233     \pgfpoint
8234     {
8235       \l_tmpa_dim + ##1 \l_tmpb_dim -
8236       \str_case:on \l_@@_hpos_block_str
8237       {
8238         l { \l_tmpb_dim + \col@sep }
8239         c { 0.5 \l_tmpb_dim }
8240         r { \col@sep }
8241       }
8242     }
8243     { \l_@@_tmpc_dim }
8244   }
8245   \pgfset { inner~sep = \c_zero_dim }
8246   \pgfnode
8247   { rectangle }
8248   {
8249     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8250     {
8251       \str_case:on \l_@@_hpos_block_str
8252       {
8253         l { north-west }
8254         c { north }
8255         r { north-east }

```

```

8256     }
8257   }
8258   {
8259     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8260     {
8261       \str_case:on \l_@@_hpos_block_str
8262       {
8263         l { south~west }
8264         c { south }
8265         r { south~east }
8266       }
8267     }
8268     {
8269       \bool_lazy_any:nTF
8270       {
8271         { \int_compare_p:nNn { #1 } = { #3 } }
8272         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8273         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8274       }
8275       {
8276         \str_case:on \l_@@_hpos_block_str
8277         {
8278           l { base~west }
8279           c { base }
8280           r { base~east }
8281         }
8282       }
8283       {
8284         \str_case:on \l_@@_hpos_block_str
8285         {
8286           l { west }
8287           c { center }
8288           r { east }
8289         }
8290       }
8291     }
8292   }
8293 }
8294 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8295 \group_end:
8296 }
8297 \endpgfpicture
8298 }

```

Now the case where there is no ampersand & in the content of the block.

```

8299   {
8300     \bool_if:NTF \l_@@_p_block_bool
8301     {

```

When the final user has used the key p, we have to compute the width.

```

8302   \pgfpicture
8303   \pgfrememberpicturepositiononpagetrue
8304   \pgf@relevantforpicturesizefalse
8305   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8306   {
8307     \@@_qpoint:n { col - #2 }
8308     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8309     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8310   }
8311   {
8312     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8313     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8314     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8315   }

```

```

8316         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8317     \endpgfpicture
8318     \hbox_set:Nn \l_@@_cell_box
8319     {
8320         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8321             { \g_tmpb_dim }
8322         \str_case:on \l_@@_hpos_block_str
8323             { c \centering r \raggedleft l \raggedright j { } }
8324         #6
8325         \end { minipage }
8326     }
8327 }
8328 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8329 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }

```

Now, we will put the label of the block. We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8330     \pgfpicture
8331     \pgfrememberpicturepositiononpagetrue
8332     \pgf@relevantforpicturesizefalse
8333     \bool_lazy_any:nTF
8334     {
8335         { \str_if_empty_p:N \l_@@_vpos_block_str }
8336         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8337         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8338         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8339     }
8340
8341     {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8341         \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the “last column”, we must put the block as if it was with the key `l`.

```

8342         \bool_if:nT \g_@@_last_col_found_bool
8343         {
8344             \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8345             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8346         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8347         \tl_set:Ne \l_tmpa_tl
8348         {
8349             \str_case:on \l_@@_vpos_block_str
8350             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8351             { } {
8352                 \str_case:on \l_@@_hpos_block_str
8353                 {
8354                     c { center }
8355                     l { west }
8356                     r { east }
8357                     j { center }
8358                 }
8359             }
8360             c {
8361                 \str_case:on \l_@@_hpos_block_str
8362                 {
8363                     c { center }
8364                     l { west }
8365                     r { east }
8366                     j { center }

```

```

8367         }
8368
8369     }
8370     T {
8371         \str_case:on \l_@@_hpos_block_str
8372         {
8373             c { north }
8374             l { north~west }
8375             r { north~east }
8376             j { north }
8377         }
8378
8379     }
8380     B {
8381         \str_case:on \l_@@_hpos_block_str
8382         {
8383             c { south }
8384             l { south~west }
8385             r { south~east }
8386             j { south }
8387         }
8388
8389     }
8390 }
8391 }
8392 \pgftransformshift
8393 {
8394     \pgfpointanchor
8395     {
8396         \@@_env: - #1 - #2 - block
8397         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8398     }
8399     { \l_tmpa_tl }
8400 }
8401 \pgfset { inner~sep = \c_zero_dim }
8402 \pgfnode
8403 { rectangle }
8404 { \l_tmpa_tl }
8405 { \box_use_drop:N \l_@@_cell_box } { } { }
8406 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8407 {
8408     \pgfextracty \l_tmpa_dim
8409     {
8410         \@@_qpoint:n
8411         {
8412             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8413             - base
8414         }
8415     }
8416     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

8417 \pgfpointanchor
8418 {
8419     \@@_env: - #1 - #2 - block
8420     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8421 }
8422 {
8423     \str_case:on \l_@@_hpos_block_str
8424     {
8425         c { center }

```

```

8426         l { west }
8427         r { east }
8428         j { center }
8429     }
8430 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8431     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8432     \pgfset { inner~sep = \c_zero_dim }
8433     \pgfnode
8434     { rectangle }
8435     {
8436         \str_case:on \l_@@_hpos_block_str
8437         {
8438             c { base }
8439             l { base~west }
8440             r { base~east }
8441             j { base }
8442         }
8443     }
8444     { \box_use_drop:N \l_@@_cell_box } { } { }
8445 }
8446 \endpgfpicture
8447 }
8448 \group_end:
8449 }
8450 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8451 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8452 {
8453     \tl_if_empty:NF \l_@@_opacity_tl
8454     {
8455         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8456         {
8457             \tl_set:Ne \l_@@_fill_tl
8458             {
8459                 [ opacity = \l_@@_opacity_tl ,
8460                 \tl_tail:o \l_@@_fill_tl
8461             }
8462         }
8463         {
8464             \tl_set:Ne \l_@@_fill_tl
8465             { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8466         }
8467     }
8468 }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8469 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8470 {
8471     \group_begin:
8472     \tl_clear:N \l_@@_draw_tl
8473     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8474     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8475     \pgfpicture
8476     \pgfrememberpicturepositiononpagetrue
8477     \pgf@relevantforpicturesizefalse

```

```

8478 \tl_if_empty:NF \l_@@_draw_tl
8479 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8480 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8481 { \CT@arc@ }
8482 { \@@_color:o \l_@@_draw_tl }
8483 }
8484 \pgfsetcornersarced
8485 {
8486 \pgfpoint
8487 { \l_@@_rounded_corners_dim }
8488 { \l_@@_rounded_corners_dim }
8489 }
8490 \@@_cut_on_hyphen:w #2 \q_stop
8491 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8492 {
8493 \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8494 {
8495 \@@_qpoint:n { row - \l_tmpa_tl }
8496 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8497 \@@_qpoint:n { col - \l_tmpb_tl }
8498 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8499 \@@_cut_on_hyphen:w #3 \q_stop
8500 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8501 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8502 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8503 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8504 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8505 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8506 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8507 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8508 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8509 \pgfpathrectanglecorners
8510 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8511 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8512 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8513 { \pgfusepathqstroke }
8514 { \pgfusepath { stroke } }
8515 }
8516 }
8517 \endpgfpicture
8518 \group_end:
8519 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8520 \keys_define:nn { nicematrix / BlockStroke }
8521 {
8522 color .tl_set:N = \l_@@_draw_tl ,
8523 draw .code:n =
8524 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8525 draw .default:n = default ,
8526 line-width .dim_set:N = \l_@@_line_width_dim ,
8527 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8528 rounded-corners .default:n = 4 pt
8529 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8530 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8531 {

```



```

8532 \group_begin:
8533 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8534 \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8535 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8536 \@@_cut_on_hyphen:w #2 \q_stop
8537 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8538 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8539 \@@_cut_on_hyphen:w #3 \q_stop
8540 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8541 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8542 \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8543 {
8544   \use:e
8545   {
8546     \@@_vline:n
8547     {
8548       position = ##1 ,
8549       start = \l_@@_tmpc_tl ,
8550       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8551       total-width = \dim_use:N \l_@@_line_width_dim
8552     }
8553   }
8554 }
8555 \group_end:
8556 }

8557 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8558 {
8559   \group_begin:
8560   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8561   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8562   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8563   \@@_cut_on_hyphen:w #2 \q_stop
8564   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8565   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8566   \@@_cut_on_hyphen:w #3 \q_stop
8567   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8568   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8569   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8570   {
8571     \use:e
8572     {
8573       \@@_hline:n
8574       {
8575         position = ##1 ,
8576         start = \l_@@_tmpd_tl ,
8577         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8578         total-width = \dim_use:N \l_@@_line_width_dim
8579       }
8580     }
8581   }
8582   \group_end:
8583 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax  $i-j$ ) and the third is the last cell of the block (with the same syntax).

```

8584 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8585 {
8586   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8587   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8588   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8589   { \@@_error:n { borders~forbidden } }

```

```

8590 {
8591   \tl_clear_new:N \l_@@_borders_tikz_tl
8592   \keys_set:no
8593     { nicematrix / OnlyForTikzInBorders }
8594   \l_@@_borders_clist
8595   \@@_cut_on_hyphen:w #2 \q_stop
8596   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8597   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8598   \@@_cut_on_hyphen:w #3 \q_stop
8599   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8600   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8601   \@@_stroke_borders_block_i:
8602 }
8603 }
8604 \hook_gput_code:nnn { begindocument } { . }
8605 {
8606   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8607   {
8608     \c_@@_pgfortikzpicture_tl
8609     \@@_stroke_borders_block_ii:
8610     \c_@@_endpgfortikzpicture_tl
8611   }
8612 }
8613 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8614 {
8615   \pgfrememberpicturepositiononpagetrue
8616   \pgf@relevantforpicturesizefalse
8617   \CT@arc@
8618   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8619   \clist_if_in:NnT \l_@@_borders_clist { right }
8620     { \@@_stroke_vertical:n \l_tmpb_tl }
8621   \clist_if_in:NnT \l_@@_borders_clist { left }
8622     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8623   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8624     { \@@_stroke_horizontal:n \l_tmpa_tl }
8625   \clist_if_in:NnT \l_@@_borders_clist { top }
8626     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8627 }
8628 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8629 {
8630   tikz .code:n =
8631     \cs_if_exist:NTF \tikzpicture
8632       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8633       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8634   tikz .value_required:n = true ,
8635   top .code:n = ,
8636   bottom .code:n = ,
8637   left .code:n = ,
8638   right .code:n = ,
8639   unknown .code:n = \@@_error:n { bad~border }
8640 }

```

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the `col` node).

```

8641 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8642 {
8643   \@@_qpoint:n \l_@@_tmpc_tl
8644   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8645   \@@_qpoint:n \l_tmpa_tl
8646   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8647   \@@_qpoint:n { #1 }
8648   \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

8649 {
8650   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8651   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8652   \pgfusepathqstroke
8653 }
8654 {
8655   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8656   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8657 }
8658 }

```

The following command is used to stroke the top border and the bottom border. The argument #1 is the number of row (in the sense of the row node).

```

8659 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8660 {
8661   \@@_qpoint:n \l_@@_tmpd_tl
8662   \clist_if_in:NnTF \l_@@_borders_clist { left }
8663   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8664   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8665   \@@_qpoint:n \l_tmpb_tl
8666   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8667   \@@_qpoint:n { #1 }
8668   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8669   {
8670     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8671     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8672     \pgfusepathqstroke
8673   }
8674   {
8675     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8676     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8677   }
8678 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

8679 \keys_define:nn { nicematrix / BlockBorders }
8680 {
8681   borders .clist_set:N = \l_@@_borders_clist ,
8682   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8683   rounded-corners .default:n = 4 pt ,
8684   line-width .dim_set:N = \l_@@_line_width_dim
8685 }

```

The following command will be used if the key tikz has been used for the command \Block.

#1 is a *list of lists* of TikZ keys used with the path.

*Example:* `{\offset=1pt,draw,red},{\offset=2pt,draw,blue}}`

which arises from a command such as :

`\Block[tikz={\offset=1pt,draw,red},tikz={\offset=2pt,draw,blue}]{2-2}{}`

The arguments #2 and #3 are the coordinates of the first cell and #4 and #5 the coordinates of the last cell of the block.

```

8686 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8687 {
8688   \begin { tikzpicture }
8689   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8690   \clist_map_inline:nn { #1 }
8691   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8692 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8693 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8694 (
8695 [
8696 xshift = \dim_use:N \l_@@_offset_dim ,
8697 yshift = - \dim_use:N \l_@@_offset_dim
8698 ]
8699 #2 -| #3
8700 )
8701 rectangle
8702 (
8703 [
8704 xshift = - \dim_use:N \l_@@_offset_dim ,
8705 yshift = \dim_use:N \l_@@_offset_dim
8706 ]
8707 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8708 ) ;
8709 }
8710 \end { tikzpicture }
8711 }
8712 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8713 \keys_define:nn { nicematrix / SpecialOffset }
8714 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8715 \cs_new_protected:Npn \@@_NullBlock:
8716 { \@@_collect_options:n { \@@_NullBlock_i: } }
8717 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8718 { }

```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (&). Of course, `&-in-blocks` must be in force.

```

8719 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8720 {
8721 \tl_gput_right:Ne \g_@@_pre_code_before_tl
8722 {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```

8723 \@@_subcellcolor:nnnnnnn
8724 {
8725 \tl_if_blank:nTF { #1 }
8726 { { \exp_not:n { #2 } } }
8727 { [ #1 ] { \exp_not:n { #2 } } }
8728 }
8729 { \int_use:N \l_@@_first_row_int } % first row of the block
8730 { \int_use:N \l_@@_first_col_int } % first column of the block
8731 { \int_use:N \l_@@_last_row_int } % last row of the block
8732 { \int_use:N \l_@@_last_col_int } % last column of the block
8733 { \int_use:N \l_@@_split_int }
8734 { \int_use:N \l_@@_split_i_int }
8735 }
8736 \ignorespaces
8737 }

8738 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8739 {
8740 \@@_color_opacity: #1

```

```

8741 \pgfpicture
8742 \pgf@relevantforpicturesizefalse
8743 \@@_qpoint:n { col - #3 }
8744 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8745 \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8746 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8747 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8748 \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8749 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8750 \@@_qpoint:n { row - #2 }
8751 \pgfpathrectanglecorners
8752 { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8753 { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8754 \pgfusepathqfill
8755 \endpgfpicture
8756 }

```

## 27 How to draw the dotted lines transparently

```

8757 \cs_set_protected:Npn \@@_renew_matrix:
8758 {
8759   \RenewDocumentEnvironment { pmatrix } { } {
8760     { \pNiceMatrix }
8761     { \endpNiceMatrix }
8762   \RenewDocumentEnvironment { vmatrix } { } {
8763     { \vNiceMatrix }
8764     { \endvNiceMatrix }
8765   \RenewDocumentEnvironment { Vmatrix } { } {
8766     { \VNiceMatrix }
8767     { \endVNiceMatrix }
8768   \RenewDocumentEnvironment { bmatrix } { } {
8769     { \bNiceMatrix }
8770     { \endbNiceMatrix }
8771   \RenewDocumentEnvironment { Bmatrix } { } {
8772     { \BNiceMatrix }
8773     { \endBNiceMatrix }
8774 }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8775 \keys_define:nn { nicematrix / Auto }
8776 {
8777   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8778   columns-type .value_required:n = true ,
8779   l .meta:n = { columns-type = l } ,
8780   r .meta:n = { columns-type = r } ,
8781   c .meta:n = { columns-type = c } ,
8782   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8783   delimiters / color .value_required:n = true ,
8784   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8785   delimiters / max-width .default:n = true ,
8786   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8787   delimiters .value_required:n = true ,
8788   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8789   rounded-corners .default:n = 4 pt
8790 }

```

```

8791 \NewDocumentCommand \AutoNiceMatrixWithDelims
8792 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8793 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8794 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8795 {

```

The group is for the protection of the keys.

```

8796 \group_begin:
8797 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8798 \use:e
8799 {
8800 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8801 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8802 [ \exp_not:o \l_tmpa_tl ]
8803 }
8804 \int_if_zero:nT { \l_@@_first_row_int }
8805 {
8806 \int_if_zero:nT { \l_@@_first_col_int } { & }
8807 \prg_replicate:nn { #4 - 1 } { & }
8808 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8809 }
8810 \prg_replicate:nn { #3 }
8811 {
8812 \int_if_zero:nT { \l_@@_first_col_int } { & }

```

We put { } before #6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).

```

8813 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8814 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8815 }
8816 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8817 {
8818 \int_if_zero:nT { \l_@@_first_col_int } { & }
8819 \prg_replicate:nn { #4 - 1 } { & }
8820 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8821 }
8822 \end { NiceArrayWithDelims }
8823 \group_end:
8824 }
8825 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8826 {
8827 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8828 {
8829 \bool_gset_true:N \g_@@_delims_bool
8830 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8831 \AutoNiceMatrixWithDelims { #2 } { #3 }
8832 }
8833 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8834 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8835 {
8836 \group_begin:
8837 \bool_gset_false:N \g_@@_delims_bool
8838 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8839 \group_end:
8840 }

```

## 29 The redefinition of the command `\dotfill`

```

8841 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8842 \cs_new_protected:Npn \@@_dotfill:
8843 {

```

First, we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in case of use of `\dotfill` “internally” in the cell (e.g. `\hbox to 1cm {\dotfill}`).

```

8844 \@@_old_dotfill:
8845 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8846 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8847 \cs_new_protected:Npn \@@_dotfill_i:
8848 {
8849   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8850   { \@@_old_dotfill: }
8851 }

```

## 30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8852 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8853 {
8854   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8855   {
8856     \@@_actually_diagbox:nnnnnn
8857     { \int_use:N \c@iRow }
8858     { \int_use:N \c@jCol }
8859     { \int_use:N \c@iRow }
8860     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8861   { \g_@@_row_style_tl \exp_not:n { #1 } }
8862   { \g_@@_row_style_tl \exp_not:n { #2 } }
8863 }

```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `corners`.

```

8864 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8865 {
8866   { \int_use:N \c@iRow }
8867   { \int_use:N \c@jCol }
8868   { \int_use:N \c@iRow }
8869   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8870   { }
8871 }
8872 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8873 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8874 {
8875   \pgfpicture
8876   \pgf@relevantforpicturesizefalse
8877   \pgfrememberpicturepositiononpagetrue
8878   \@@_qpoint:n { row - #1 }
8879   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8880   \@@_qpoint:n { col - #2 }
8881   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8882   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8883   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8884   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8885   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8886   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8887   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8888   {

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. The package `nicematrix` uses it even if `colortbl` is not loaded.

```

8889   \CT@arc@
8890   \pgfsetroundcap
8891   \pgfusepathqstroke
8892 }
8893 \pgfset { inner~sep = 1 pt }
8894 \pgfscope
8895 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8896 \pgfnode { rectangle } { south~west }
8897 {
8898   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8899   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8900   \end { minipage }
8901 }
8902 { }
8903 { }
8904 \endpgfscope
8905 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8906 \pgfnode { rectangle } { north~east }
8907 {
8908   \begin { minipage } { 20 cm }
8909   \raggedleft
8910   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8911   \end { minipage }
8912 }
8913 { }
8914 { }
8915 \endpgfpicture
8916 }

```

## 31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 90.



In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8917 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }
```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```
8918 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }
```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```
8919 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8920 {
8921   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8922   \@@_CodeAfter_iv:n
8923 }
```

We catch the argument of the command `\end` (in `#1`).

```
8924 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8925 {
```

If this is really the end of the current environment (of `nicematrix`), we put back the command `\end` and its argument in the TeX flow.

```
8926   \str_if_eq:eeTF { \@currenvir } { #1 }
8927   { \end { #1 } }
```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```
8928   {
8929     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8930     \@@_CodeAfter_ii:n
8931   }
8932 }
```

## 32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8933 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8934 {
8935   \pgfpicture
8936   \pgfrememberpicturepositiononpagetrue
8937   \pgf@relevantforpicturesizefalse
```

`\l_@@_y_initial_dim` and `\l_@@_y_final_dim` will be the  $y$ -values of the extremities of the delimiter we will have to construct.

```
8938   \@@_qpoint:n { row - 1 }
8939   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8940   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8941   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
```

We will compute in `\l_tmpa_dim` the  $x$ -value where we will have to put our delimiter (on the left side or on the right side).

```

8942   \bool_if:nTF { #3 }
8943   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8944   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8945   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8946   {
8947     \cs_if_exist:cT
8948     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8949     {
8950       \pgfpointanchor
8951       { \@@_env: - ##1 - #2 }
8952       { \bool_if:nTF { #3 } { west } { east } }
8953       \dim_set:Nn \l_tmpa_dim
8954       {
8955         \bool_if:nTF { #3 }
8956         { \dim_min:nn }
8957         { \dim_max:nn }
8958         \l_tmpa_dim
8959         { \pgf@x }
8960       }
8961     }
8962   }

```

Now we can put the delimiter with a node of PGF.

```

8963   \pgfset { inner~sep = \c_zero_dim }
8964   \dim_zero:N \nulldelimiterspace
8965   \pgftransformshift
8966   {
8967     \pgfpoint
8968     { \l_tmpa_dim }
8969     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8970   }
8971   \pgfnode
8972   { rectangle }
8973   { \bool_if:nTF { #3 } { east } { west } }
8974   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8975     \nullfont
8976     $ % $
8977     \@@_color:o \l_@@_delimiters_color_tl
8978     \bool_if:nTF { #3 } { \left #1 } { \left . }
8979     \vcenter
8980     {
8981       \nullfont
8982       \hrule \@@height
8983       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8984       \@depth \c_zero_dim
8985       \@width \c_zero_dim
8986     }
8987     \bool_if:nTF { #3 } { \right . } { \right #1 }
8988     $ % $
8989   }
8990   { }
8991   { }
8992   \endpgfpicture
8993 }

```

### 33 The command `\SubMatrix`

```

8994 \keys_define:nn { nicematrix / sub-matrix }
8995 {
8996   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8997   extra-height .value_required:n = true ,
8998   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8999   left-xshift .value_required:n = true ,
9000   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9001   right-xshift .value_required:n = true ,
9002   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9003   xshift .value_required:n = true ,
9004   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9005   delimiters / color .value_required:n = true ,
9006   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9007   slim .default:n = true ,
9008   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9009   hlines .default:n = all ,
9010   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9011   vlines .default:n = all ,
9012   hvlines .meta:n = { hlines, vlines } ,
9013   hvlines .value_forbidden:n = true
9014 }
9015 \keys_define:nn { nicematrix }
9016 {
9017   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9018   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9019   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9020   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9021 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9022 \keys_define:nn { nicematrix / SubMatrix }
9023 {
9024   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9025   delimiters / color .value_required:n = true ,
9026   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9027   hlines .default:n = all ,
9028   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9029   vlines .default:n = all ,
9030   hvlines .meta:n = { hlines, vlines } ,
9031   hvlines .value_forbidden:n = true ,
9032   name .code:n =
9033     \tl_if_empty:nTF { #1 }
9034     { \@@_error:n { Invalid-name } }
9035     {
9036       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9037       {
9038         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9039         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9040         {
9041           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9042           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9043         }
9044       }
9045       { \@@_error:n { Invalid-name } }
9046     } ,
9047   name .value_required:n = true ,
9048   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9049   rules .value_required:n = true ,
9050   code .tl_set:N = \l_@@_code_tl ,
9051   code .value_required:n = true ,
9052   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9053 }

```

```

9054 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
9055 {
9056   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9057   {
9058     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9059     [
9060       delimiters / color = \l_@@_delimiters_color_tl ,
9061       hlines = \l_@@_submatrix_hlines_clist ,
9062       vlines = \l_@@_submatrix_vlines_clist ,
9063       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9064       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9065       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9066       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9067       #5
9068     ]
9069   }
9070   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9071   \ignorespaces
9072 }
9073 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9074 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9075 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9076 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9077 {
9078   \seq_gput_right:Ne \g_@@_submatrix_seq
9079   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9080   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9081   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9082   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9083   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9084 }
9085 }

```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

9086 \NewDocumentCommand \@@_compute_i_j:nn
9087 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9088 { \@@_compute_i_j:nnnn #1 #2 }
9089 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9090 {
9091   \def \l_@@_first_i_tl { #1 }
9092   \def \l_@@_first_j_tl { #2 }
9093   \def \l_@@_last_i_tl { #3 }
9094   \def \l_@@_last_j_tl { #4 }
9095   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9096   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9097   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9098   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9099   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9100   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9101   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9102   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9103 }

```

In the pre-code-after and in the `\CodeAfter` the following command `\@@_SubMatrix` will be linked to `\SubMatrix`.

- #1 is the left delimiter;
- #2 is the upper-left cell of the matrix with the format  $i$ - $j$ ;

- #3 is the lower-right cell of the matrix with the format  $i$ - $j$ ;
- #4 is the right delimiter;
- #5 is the list of options of the command;
- #6 is the potential subscript;
- #7 is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command `\Cdots`.

```

9104 \hook_gput_code:nnn { begindocument } { . }
9105   {
9106     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9107     \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9108       { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9109   }
9110 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9111   {
9112     \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9113 \@@_compute_i_j:nn { #2 } { #3 }
9114 \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
9115   { \def \arraystretch { 1 } }
9116 \bool_lazy_or:nnTF
9117   { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9118   { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9119   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9120   {
9121     \str_clear_new:N \l_@@_submatrix_name_str
9122     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9123     \pgfpicture
9124     \pgfrememberpicturepositiononpagetrue
9125     \pgf@relevantforpicturesizefalse
9126     \pgfset { inner~sep = \c_zero_dim }
9127     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9128     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9129 \bool_if:NTF \l_@@_submatrix_slim_bool
9130   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
9131   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
9132   {
9133     \cs_if_exist:cT
9134       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9135     {
9136       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9137       \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9138       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9139     }
9140     \cs_if_exist:cT
9141       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9142     {
9143       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9144       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9145       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9146     }
9147   }
9148 \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
9149   { \@@_error:nn { Impossible~delimiter } { left } }
9150   {
9151     \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

9152         { \@@_error:nn { Impossible-delimiter } { right } }
9153         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9154     }
9155     \endpgfpicture
9156 }
9157 \group_end:
9158 \ignorespaces
9159 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9160 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9161 {
9162     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9163     \dim_set:Nn \l_@@_y_initial_dim
9164     {
9165         \fp_to_dim:n
9166         {
9167             \pgf@y
9168             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9169         }
9170     }
9171     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9172     \dim_set:Nn \l_@@_y_final_dim
9173     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9174     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9175     {
9176         \cs_if_exist:cT
9177         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9178         {
9179             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9180             \dim_set:Nn \l_@@_y_initial_dim
9181             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9182         }
9183         \cs_if_exist:cT
9184         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9185         {
9186             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9187             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9188             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9189         }
9190     }
9191     \dim_set:Nn \l_tmpa_dim
9192     {
9193         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9194         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9195     }
9196     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9197     \group_begin:
9198     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9199     \@@_set_CTarc:o \l_@@_rules_color_tl
9200     \CT@arc@

```

Now, we draw the potential vertical rules specified in the preamble of the environments with the letter fixed with the key `vlines-in-sub-matrix`. The list of the columns where there is such rule to draw is in `\g_@@_cols_vlism_seq`.

```

9201     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9202     {
9203         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9204         {
9205             \int_compare:nNnT
9206             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```
9207 {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
9208 \@@_qpoint:n { col - ##1 }
9209 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9210 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9211 \pgfusepathqstroke
9212 }
9213 }
9214 }
```

Now, we draw the vertical rules specified in the key `vlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```
9215 \str_if_eq:eeTF { \l_@@_submatrix_vlines_clist } { all }
9216 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9217 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9218 {
9219   \bool_lazy_and:nnTF
9220   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9221   {
9222     \int_compare_p:nNn
9223     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9224   {
9225     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9226     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9227     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9228     \pgfusepathqstroke
9229   }
9230   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9231 }
```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```
9232 \str_if_eq:eeTF { \l_@@_submatrix_hlines_clist } { all }
9233 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9234 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9235 {
9236   \bool_lazy_and:nnTF
9237   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9238   {
9239     \int_compare_p:nNn
9240     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9241   {
9242     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
9243 \group_begin:
```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```
9244 \dim_set:Nn \l_tmpa_dim
9245 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9246 \str_case:nn { #1 }
9247 {
9248   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9249   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9250   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9251   }
9252   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```
9253 \dim_set:Nn \l_tmpb_dim
9254 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9255 \str_case:nn { #2 }
9256 {
9257   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
```

```

9258         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9259         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9260     }
9261     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9262     \pgfusepathqstroke
9263     \group_end:
9264 }
9265 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9266 }

```

If the key name has been used for the command `\SubMatrix`, we create a PGF node with that name for the submatrix (this node does not encompass the delimiters that we will put after).

```

9267 \str_if_empty:NF \l_@@_submatrix_name_str
9268 {
9269     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9270     \l_@@_x_initial_dim \l_@@_y_initial_dim
9271     \l_@@_x_final_dim \l_@@_y_final_dim
9272 }
9273 \group_end:

```

The group was for `\CT@arc@` (the color of the rules).

Now, we deal with the left delimiter. Of course, the environment `{pgfscope}` is for the `\pgftransformshift`.

```

9274 \begin { pgfscope }
9275 \pgftransformshift
9276 {
9277     \pgfpoint
9278     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9279     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9280 }
9281 \str_if_empty:NTF \l_@@_submatrix_name_str
9282 { \@@_node_left:nn #1 { } }
9283 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9284 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9285 \pgftransformshift
9286 {
9287     \pgfpoint
9288     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9289     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9290 }
9291 \str_if_empty:NTF \l_@@_submatrix_name_str
9292 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9293 {
9294     \@@_node_right:nnnn #2
9295     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9296 }

```

Now, we deal with the key code of `\SubMatrix`. That key should contain a TikZ instruction and the nodes in that instruction will be relative to the current `\SubMatrix`. That's why we need a redefinition of `\pgfpointanchor`.

```

9297 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9298 \flag_clear_new:N \l_@@_code_flag
9299 \l_@@_code_tl
9300 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ ,  $\text{row-}i$ ,  $\text{col-}j$  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9301 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```



The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
9302 \cs_new:Npn \@@_pgfpointanchor:n #1
9303 { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```
9304 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9305 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9306 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9307 { }
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9308 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9309 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9310 { \@@_pgfpointanchor_ii:n { #1 } }
9311 }
```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```
9312 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9313 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
9314 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9315 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9316 { }
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9317 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9318 { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
9319 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9320 }
```

The following function is for the case when the name contains an hyphen.

```
9321 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9322 { }
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9323 \@@_env:
9324 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9325 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9326 }
```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

9327 \tl_const:Nn \c_@@_integers_alist_tl
9328 {
9329   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9330   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9331   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9332   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9333 }

9334 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9335 {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form  $i-j$ . That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises with its command `\name_of_command` (see above).

In that case, the  $i$  of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the  $j$  arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

9336   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9337   {
9338     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9339   \@@_env: -
9340   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9341   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9342   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9343 }
9344 {
9345   \str_if_eq:eeTF { #1 } { last }
9346   {
9347     \flag_raise:N \l_@@_code_flag
9348     \@@_env: -
9349     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9350     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9351     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9352   }
9353   { #1 }
9354 }
9355 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

9356 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9357 {
9358   \pgfnode
9359   { rectangle }
9360   { east }
9361   {
9362     \nullfont
9363     $ % $
9364     \@@_color:o \l_@@_delimiters_color_tl
9365     \left #1
9366     \vcenter
9367     {
9368       \nullfont
9369       \hrule \@height \l_tmpa_dim
9370       \@depth \c_zero_dim

```

```

9371         \@width \c_zero_dim
9372     }
9373     \right .
9374     $ % $
9375 }
9376 { #2 }
9377 { }
9378 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```

9379 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9380 {
9381     \pgfnode
9382     { rectangle }
9383     { west }
9384     {
9385         \nullfont
9386         $ % $
9387         \colorlet { current-color } { . }
9388         \@@_color:o \l_@@_delimiters_color_tl
9389         \left .
9390         \vcenter
9391         {
9392             \nullfont
9393             \hrule \@height \l_tmpa_dim
9394                 \@depth \c_zero_dim
9395                 \@width \c_zero_dim
9396         }
9397         \right #1
9398         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9399         ^ { \color { current-color } \smash { #4 } }
9400         $ % $
9401     }
9402     { #2 }
9403     { }
9404 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9405 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9406 {
9407     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9408     \ignorespaces
9409 }
9410 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9411 {
9412     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9413     \ignorespaces
9414 }
9415 \keys_define:nn { nicematrix / Brace }
9416 {
9417     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9418     left-shorten .default:n = true ,
9419     left-shorten .value_forbidden:n = true ,

```

```

9420 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9421 right-shorten .default:n = true ,
9422 right-shorten .value_forbidden:n = true ,
9423 shorten .meta:n = { left-shorten , right-shorten } ,
9424 shorten .value_forbidden:n = true ,
9425 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9426 yshift .value_required:n = true ,
9427 yshift .initial:n = \c_zero_dim ,
9428 color .tl_set:N = \l_tmpa_tl ,
9429 color .value_required:n = true ,
9430 unknown .code:n =
9431   \@@_unknown_key:nn
9432   { nicematrix / Brace }
9433   { Unknown-key-for-Brace }
9434 }

```

#1 is the first cell of the rectangle (with the syntax  $i-j$ ; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to *under* or *over*.

```

9435 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9436 {
9437   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9438 \@@_compute_i_j:nn { #1 } { #2 }
9439 \bool_lazy_or:nnTF
9440 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9441 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9442 {
9443   \str_if_eq:eeTF { #5 } { under }
9444   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9445   { \@@_error:nn { Construct-too-large } { \OverBrace } }
9446 }
9447 {
9448   \tl_clear:N \l_tmpa_tl
9449   \keys_set:nn { nicematrix / Brace } { #4 }
9450   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9451   \pgfpicture
9452   \pgfrememberpicturepositiononpagetrue
9453   \pgf@relevantforpicturesizefalse
9454   \bool_if:NT \l_@@_brace_left_shorten_bool
9455   {
9456     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9457     \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9458     {
9459       \cs_if_exist:cT
9460       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9461       {
9462         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9463
9464         \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9465         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9466       }
9467     }
9468   }
9469   \bool_lazy_or:nnT
9470   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9471   { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9472   {
9473     \@@_qpoint:n { col - \l_@@_first_j_tl }
9474     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9475   }
9476   \bool_if:NT \l_@@_brace_right_shorten_bool
9477   {

```

```

9478 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9479 \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9480 {
9481   \cs_if_exist:cT
9482   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9483   {
9484     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9485     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9486     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9487   }
9488 }
9489 }
9490 \bool_lazy_or:nnT
9491 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9492 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9493 {
9494   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9495   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9496 }
9497 \pgfset { inner~sep = \c_zero_dim }
9498 \str_if_eq:eeTF { #5 } { under }
9499 { \@@_underbrace_i:n { #3 } }
9500 { \@@_overbrace_i:n { #3 } }
9501 \endpgfpicture
9502 }
9503 \group_end:
9504 }

```

The argument is the text to put above the brace.

```

9505 \cs_new_protected:Npn \@@_overbrace_i:n #1
9506 {
9507   \@@_qpoint:n { row - \l_@@_first_i_tl }
9508   \pgftransformshift
9509   {
9510     \pgfpoint
9511     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9512     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9513   }
9514   \pgfnode
9515   { rectangle }
9516   { south }
9517   {
9518     \vtop
9519     {
9520       \group_begin:
9521       \everycr { }
9522       \halign
9523       {
9524         \hfil ## \hfil \crcr
9525         \bool_if:NTF \l_@@_tabular_bool
9526         { \begin { tabular } { c } #1 \end { tabular } }
9527         { $ \begin { array } { c } #1 \end { array } $ }
9528         \cr
9529         $ % $
9530         \overbrace
9531         {
9532           \hbox_to_wd:nn
9533           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9534           { }
9535         }
9536         $ % $
9537         \cr
9538       }
9539       \group_end:

```

```

9540     }
9541   }
9542   { }
9543   { }
9544 }

```

The argument is the text to put under the brace.

```

9545 \cs_new_protected:Npn \@@_underbrace_i:n #1
9546 {
9547   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9548   \pgftransformshift
9549   {
9550     \pgfpoint
9551     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9552     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9553   }
9554   \pgfnode
9555   { rectangle }
9556   { north }
9557   {
9558     \group_begin:
9559     \everycr { }
9560     \vbox
9561     {
9562       \halign
9563       {
9564         \hfil ## \hfil \crcr
9565         $ % $
9566         \underbrace
9567         {
9568           \hbox_to_wd:nn
9569           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9570           { }
9571         }
9572         $ % $
9573         \cr
9574         \bool_if:NTF \l_@@_tabular_bool
9575         { \begin { tabular } { c } #1 \end { tabular } }
9576         { $ \begin { array } { c } #1 \end { array } $ }
9577         \cr
9578       }
9579     }
9580     \group_end:
9581   }
9582   { }
9583   { }
9584 }

```

## 35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9585 \AddToHook { package / tikz / after }
9586 {

```

```

9587 \tikzset
9588 {
9589     nicematrix / brace / .style =
9590     {
9591         decoration = { brace , raise = -0.15 em } ,
9592         decorate ,
9593     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9594     nicematrix / mirrored-brace / .style =
9595     {
9596         nicematrix / brace ,
9597         decoration = mirror ,
9598     }
9599 }
9600 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9601 \keys_define:nn { nicematrix / Hbrace }
9602 {
9603     color .code:n = ,
9604     horizontal-label .code:n = ,
9605     horizontal-labels .code:n = ,
9606     shorten .code:n = ,
9607     shorten-start .code:n = ,
9608     shorten-end .code:n = ,
9609     shorten+ .code:n = ,
9610     shorten-start+ .code:n = ,
9611     shorten-end+ .code:n = ,
9612     shorten~+ .code:n = ,
9613     shorten-start~+ .code:n = ,
9614     shorten-end~+ .code:n = ,
9615     brace-shift .code:n = ,
9616     brace-shift+ .code:n = ,
9617     brace-shift~+ .code:n = ,
9618     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9619 }

```

Here we need an “fully expandable” command.

```

9620 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9621 {
9622     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9623     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9624     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9625 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9626 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9627 {
9628     \int_compare:nNnTF { \c@iRow } < { 2 }
9629     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9630     \str_if_eq:nnTF { #2 } { * }
9631     {
9632         \bool_set_true:N \l_@@_nullify_dots_bool
9633         \Ldots
9634         [
9635             line-style = nicematrix / brace ,
9636             #1 ,

```

```

9637         up =
9638         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9639     ]
9640 }
9641 {
9642     \Hdotsfor
9643     [
9644         line-style = nicematrix / brace ,
9645         #1 ,
9646         up =
9647         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9648     ]
9649     { #2 }
9650 }
9651 }
9652 {
9653     \str_if_eq:nnTF { #2 } { * }
9654     {
9655         \bool_set_true:N \l_@@_nullify_dots_bool
9656         \Ldots
9657         [
9658             line-style = nicematrix / mirrored-brace ,
9659             #1 ,
9660             down =
9661             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9662         ]
9663     }
9664     {
9665         \Hdotsfor
9666         [
9667             line-style = nicematrix / mirrored-brace ,
9668             #1 ,
9669             down =
9670             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9671         ]
9672         { #2 }
9673     }
9674 }
9675 \keys_set:nn { nicematrix / Hbrace } { #1 }
9676 }

9677 \NewDocumentCommand { \@@_Vbrace } { 0 { } } m m }
9678 {
9679     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9680     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9681     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9682 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9683 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9684 {
9685     \int_compare:nNnTF { \c@jCol } < { 2 }
9686     {
9687         \str_if_eq:nnTF { #2 } { * }
9688         {
9689             \bool_set_true:N \l_@@_nullify_dots_bool
9690             \Vdots
9691             [
9692                 Vbrace ,
9693                 line-style = nicematrix / mirrored-brace ,
9694                 #1 ,
9695                 down =

```



```

9696         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9697     ]
9698 }
9699 {
9700     \Vdotsfor
9701     [
9702         Vbrace ,
9703         line-style = nicematrix / mirrored-brace ,
9704         #1 ,
9705         down =
9706         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9707     ]
9708     { #2 }
9709 }
9710 }
9711 {
9712     \str_if_eq:nnTF { #2 } { * }
9713     {
9714         \bool_set_true:N \l_@@_nullify_dots_bool
9715         \Vdots
9716         [
9717             Vbrace ,
9718             line-style = nicematrix / brace ,
9719             #1 ,
9720             up =
9721             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9722         ]
9723     }
9724     {
9725         \Vdotsfor
9726         [
9727             Vbrace ,
9728             line-style = nicematrix / brace ,
9729             #1 ,
9730             up =
9731             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9732         ]
9733         { #2 }
9734     }
9735 }
9736 \keys_set:nn { nicematrix / Hbrace } { #1 }
9737 }

```

## 36 The command TikzEveryCell

```

9738 \bool_new:N \l_@@_not_empty_bool
9739 \bool_new:N \l_@@_empty_bool
9740
9741 \keys_define:nn { nicematrix / TikzEveryCell }
9742 {
9743     not-empty .code:n =
9744         \bool_lazy_or:nnTF
9745         { \l_@@_in_code_after_bool }
9746         { \g_@@_create_cell_nodes_bool }
9747         { \bool_set_true:N \l_@@_not_empty_bool }
9748         { @@_error:n { detection-of-empty-cells } } ,
9749     not-empty .value_forbidden:n = true ,
9750     empty .code:n =
9751         \bool_lazy_or:nnTF

```

```

9752     { \l_@@_in_code_after_bool }
9753     { \g_@@_create_cell_nodes_bool }
9754     { \bool_set_true:N \l_@@_empty_bool }
9755     { \@@_error:n { detection-of-empty-cells } } } ,
9756     empty .value_forbidden:n = true ,
9757     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9758 }
9759
9760
9761 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9762 {
9763     \IfPackageLoadedTF { tikz }
9764     {
9765         \group_begin:
9766         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9767         \tl_set:Nn \l_tmpa_tl { { #2 } }
9768         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9769         { \@@_for_a_block:nnnnn ##1 }
9770         \@@_all_the_cells:
9771         \group_end:
9772     }
9773     { \@@_error:n { TikzEveryCell-without-tikz } }
9774 }
9775
9776
9777 \cs_new_protected:Nn \@@_all_the_cells:
9778 {
9779     \int_step_inline:nn \c@iRow
9780     {
9781         \int_step_inline:nn \c@jCol
9782         {
9783             \cs_if_exist:cF { cell - ##1 - ####1 }
9784             {
9785                 \clist_if_in:NcF \l_@@_corners_cells_clist
9786                 { ##1 - ####1 }
9787                 {
9788                     \bool_set_false:N \l_tmpa_bool
9789                     \cs_if_exist:cTF
9790                     { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
9791                     {
9792                         \bool_if:NF \l_@@_empty_bool
9793                         { \bool_set_true:N \l_tmpa_bool }
9794                     }
9795                     {
9796                         \bool_if:NF \l_@@_not_empty_bool
9797                         { \bool_set_true:N \l_tmpa_bool }
9798                     }
9799                     \bool_if:NT \l_tmpa_bool
9800                     {
9801                         \@@_block_tikz:nnnnn
9802                         \l_tmpa_tl { ##1 } { ####1 } { ##1 } { ####1 }
9803                     }
9804                 }
9805             }
9806         }
9807     }
9808 }
9809
9810 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9811 {
9812     \bool_if:NF \l_@@_empty_bool

```

```

9813 {
9814   \@@_block_tikz:nnnn
9815   \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9816 }
9817 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9818 }
9819
9820 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9821 {
9822   \int_step_inline:nnn { #1 } { #3 }
9823   {
9824     \int_step_inline:nnn { #2 } { #4 }
9825     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9826   }
9827 }

```

### 37 The command \CreateBlocksInColumn

```

9828 \cs_new_protected:Npn \@@_create_blocks_in_col:n #1
9829 { \clist_gput_right:Nn \g_@@_cbic_clist { #1 } }
9830 \cs_new_protected:Npn \@@_cbic:
9831 {
9832   \clist_map_inline:Nn \g_@@_cbic_clist
9833   {
9834     \cs_set:cpn
9835     {
9836       pgf @ sh @ ns @ \@@_env:
9837       - \int_eval:n { \c@iRow + 1 } - ##1 }
9838     { rien }
9839
9840     \l_tmpa_int will be the first row of the block being created.
9841     \int_set:Nn \l_tmpa_int { 1 }
9842     \int_step_inline:nn { \c@iRow + 1 }
9843     {
9844       \cs_if_exist:cT
9845       { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
9846       {
9847         \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
9848         {
9849           \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9850           {
9851             { \int_use:N \l_tmpa_int }
9852             { ##1 }
9853             { \int_eval:n { #####1 - 1 } }
9854             { ##1 }
9855           }
9856           \int_set:Nn \l_tmpa_int { #####1 }
9857         }
9858       }
9859     }
9860     \clist_gclear:N \g_@@_cbic_clist
9861   }

```

### 38 The command \ShowCellNames

```

9862 \NewDocumentCommand \@@_ShowCellNames { }
9863 {
9864   \bool_if:NT \l_@@_in_code_after_bool

```

```

9865 {
9866   \pgfpicture
9867   \pgfrememberpicturepositiononpagetrue
9868   \pgf@relevantforpicturesizefalse
9869   \pgfpathrectanglecorners
9870   { \@@_qpoint:n { 1 } }
9871   {
9872     \@@_qpoint:n
9873     { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9874   }
9875   \pgfsetfillopacity { 0.75 }
9876   \pgfsetfillcolor { white }
9877   \pgfusepathqfill
9878   \endpgfpicture
9879 }
9880 \dim_gzero_new:N \g_@@_tmpc_dim
9881 \dim_gzero_new:N \g_@@_tmpd_dim
9882 \dim_gzero_new:N \g_@@_tmpe_dim
9883 \int_step_inline:nn { \c@iRow }
9884 {
9885   \bool_if:NTF \l_@@_in_code_after_bool
9886   {
9887     \pgfpicture
9888     \pgfrememberpicturepositiononpagetrue
9889     \pgf@relevantforpicturesizefalse
9890   }
9891   { \begin { pgfpicture } }
9892   \@@_qpoint:n { row - ##1 }
9893   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9894   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9895   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9896   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9897   \bool_if:NTF \l_@@_in_code_after_bool
9898   { \endpgfpicture }
9899   { \end { pgfpicture } }
9900   \int_step_inline:nn { \c@jCol }
9901   {
9902     \hbox_set:Nn \l_tmpa_box
9903     {
9904       \normalfont \Large \sffamily \bfseries
9905       \bool_if:NTF \l_@@_in_code_after_bool
9906       { \color { red } }
9907       { \color { red ! 50 } }
9908       ##1 - ####1
9909     }
9910     \bool_if:NTF \l_@@_in_code_after_bool
9911     {
9912       \pgfpicture
9913       \pgfrememberpicturepositiononpagetrue
9914       \pgf@relevantforpicturesizefalse
9915     }
9916     { \begin { pgfpicture } }
9917     \@@_qpoint:n { col - ####1 }
9918     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9919     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9920     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9921     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9922     \bool_if:NTF \l_@@_in_code_after_bool
9923     { \endpgfpicture }
9924     { \end { pgfpicture } }
9925     \fp_set:Nn \l_tmpa_fp
9926     {
9927       \fp_min:nn

```

```

9928         {
9929             \fp_min:nn
9930             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9931             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9932         }
9933         { 1.0 }
9934     }
9935     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9936     \pgfpicture
9937     \pgfrememberpicturepositiononpagetrue
9938     \pgf@relevantforpicturesizefalse
9939     \pgftransformshift
9940     {
9941         \pgfpoint
9942         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9943         { \dim_use:N \g_tmpa_dim }
9944     }
9945     \pgfnode
9946     { rectangle }
9947     { center }
9948     { \box_use:N \l_tmpa_box }
9949     { }
9950     { }
9951     \endpgfpicture
9952 }
9953 }
9954 }

```

## 39 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9955 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to true if the option `footnotehyper` is used.

```

9956 \bool_new:N \g_@@_footnote_bool
9957 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9958 {
9959     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9960     but-that-key-is-unknown. \\
9961     It-will-be-ignored. \\
9962     For-a-list-of-the-available-keys,~type-H~<return>.
9963 }
9964 {
9965     The-available-keys-are~(in-alphabetic-order):~
9966     footnote,~
9967     footnotehyper,~
9968     messages-for-Overleaf,~
9969     renew-dots~and~
9970     renew-matrix.
9971 }
9972 \keys_define:nn { nicematrix }
9973 {
9974     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

9975     renew-dots .value_forbidden:n = true ,
9976     renew-matrix .code:n = \@_renew_matrix: ,
9977     renew-matrix .value_forbidden:n = true ,
9978     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9979     footnote .bool_set:N = \g_@@_footnote_bool ,
9980     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9981     unknown .code:n = \@_error:n { Unknown-key-for-package }
9982   }
9983 \ProcessKeyOptions

9984 \@_msg_new:nn { footnote-with-footnotehyper-package }
9985 {
9986   You-can't-use-the-option-'footnote'~because-the-package~
9987   footnotehyper-has-already-been-loaded.~
9988   If-you-want,-you-can-use-the-option-'footnotehyper'~and-the-footnotes~
9989   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9990   of-the-package-footnotehyper.\\
9991   The-package-footnote-won't-be-loaded.
9992 }

9993 \@_msg_new:nn { footnotehyper-with-footnote-package }
9994 {
9995   You-can't-use-the-option-'footnotehyper'~because-the-package~
9996   footnote-has-already-been-loaded.~
9997   If-you-want,-you-can-use-the-option-'footnote'~and-the-footnotes~
9998   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9999   of-the-package-footnote.\\
10000   The-package-footnotehyper-won't-be-loaded.
10001 }

10002 \bool_if:NT \g_@@_footnote_bool
10003 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10004   \IfClassLoadedTF { beamer }
10005   { \bool_set_false:N \g_@@_footnote_bool }
10006   {
10007     \IfPackageLoadedTF { footnotehyper }
10008     { \@_error:n { footnote-with-footnotehyper-package } }
10009     { \usepackage { footnote } }
10010   }
10011 }

10012 \bool_if:NT \g_@@_footnotehyper_bool
10013 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10014   \IfClassLoadedTF { beamer }
10015   { \bool_set_false:N \g_@@_footnote_bool }
10016   {
10017     \IfPackageLoadedTF { footnote }
10018     { \@_error:n { footnotehyper-with-footnote-package } }
10019     { \usepackage { footnotehyper } }
10020   }
10021   \bool_set_true:N \g_@@_footnote_bool
10022 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

## 40 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```
10023 \bool_new:N \l_@@_underscore_loaded_bool
10024 \IfPackageLoadedT { underscore }
10025   { \bool_set_true:N \l_@@_underscore_loaded_bool }
10026 \hook_gput_code:nnn { begindocument } { . }
10027 {
10028   \bool_if:NF \l_@@_underscore_loaded_bool
10029   {
10030     \IfPackageLoadedT { underscore }
10031     { \@@_error:n { underscore-after-nicematrix } }
10032   }
10033 }
```

## 41 Compatibility with threeparttable

```
10034 \hook_gput_code:nnn { begindocument } { . }
10035 {
10036   \IfPackageLoadedT { threeparttable }
10037   {
10038     \AddToHook { env / threeparttable / begin }
10039     {
10040       \TPT@hookin { NiceTabular }
10041       \TPT@hookin { NiceTabular* }
10042       \TPT@hookin { NiceTabularX }
10043     }
10044   }
10045 }
```

## 42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is +.

**#1** is a clist of names of sets of keys and **#2** is the error message to send.

```
10046 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10047 {
10048   \str_if_eq:eeTF
10049   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10050   { + }
10051   {
10052     \str_set:Ne \l_tmpa_str
10053     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10054     \bool_set_false:N \l_tmpa_bool
10055     \clist_map_inline:nn { #1 }
10056     {
10057       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10058       {
10059         \@@_error:n { key~without~+~exists }
10060         \bool_set_true:N \l_tmpa_bool
10061         \clist_map_break:
10062       }
10063     }
10064   }
10065 }
```

```

10063     }
10064     \bool_if:NF \l_tmpa_bool
10065     {
10066         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10067         \@@_unknown_key_i:nn { #1 } { #2 }
10068     }
10069 }
10070 { \@@_unknown_key_i:nn { #1 } { #2 } }
10071 }

```

We try a normalisation of the name of the key, and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of nicematrix).

#1 is a clist of names of sets of keys and #2 is the error message to send.

```

10072 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10073 {
10074     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10075     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10076     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10077     \bool_set_false:N \l_tmpa_bool
10078     \clist_map_inline:nn { #1 }
10079     {
10080         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10081         {
10082             \@@_error:n { key~with~normal~form~exists }
10083             \bool_set_true:N \l_tmpa_bool
10084             \clist_map_break:
10085         }
10086     }
10087     \bool_if:NF \l_tmpa_bool
10088     {
10089         \@@_error:n { #2 }

```

If `messages-for-Overleaf` is not in force, the list of the available keys is not written in the main error message but only in the complement (if the final user presses H). That's why we write, in all circumstances, the list of the available keys in order to facilitate the work of the systems which analyze the error by IA (such as Prism).

```

10090     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10091     { \msg_info:nn { nicematrix } { #2~+ } }
10092 }
10093 }
10094 \@@_msg_new:nn { key~without~+~exists }
10095 {
10096     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10097     additive~syntax~(with~+=).\
10098     It~will~be~ignored.\
10099 }
10100 \@@_msg_new:nn { key~with~normal~form~exists }
10101 {
10102     The~key~'\l_keys_key_str'~does~not~exists.\
10103     It~will~be~ignored.\
10104     Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10105 }
10106 \str_const:Ne \c_@@_available_keys_str
10107 {
10108     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10109     { For~a~list~of~the~available~keys,~type~H~<return>. }
10110 }

```



```

10111 \seq_new:N \g_@@_types_of_matrix_seq
10112 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10113 {
10114     NiceMatrix ,
10115     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10116 }
10117 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10118 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

10119 \cs_new_protected:Npn \@@_err_too_many_cols:
10120 {
10121     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10122     { \@@_fatal:nn { too-many-cols-for-array } }
10123     \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
10124     { \@@_fatal:n { too-many-cols-for-matrix } }
10125     \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
10126     { \@@_fatal:n { too-many-cols-for-matrix } }
10127     \bool_if:NF \l_@@_last_col_without_value_bool
10128     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10129 }

```

The following command must *not* be protected since it's used in an error message.

```

10130 \cs_new:Npn \@@_message_hdotsfor:
10131 {
10132     \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
10133     { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10134       \token_to_str:N \Hbrace \ is~incorrect. }
10135 }

10136 \cs_new_protected:Npn \@@_Hline_in_cell:
10137 { \@@_fatal:n { Misuse-of-Hline } }

10138 \@@_msg_new:nn { Misuse-of-Hline }
10139 {
10140     Misuse-of-Hline. \\
10141     Error-in-your-row~ \int_eval:n { \c@iRow }. \\
10142     \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must-be-used-only~
10143     at~the~beginning-of-a~row.\\
10144     That~error~is~fatal.
10145 }

10146 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10147 {
10148     Incompatible-options.\\
10149     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
10150     The-output-will-not-be-reliable.
10151 }

10152 \@@_msg_new:nn { Body-alone }
10153 {
10154     \token_to_str:N \Body\ alone. \\
10155     You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10156     That~error~is~fatal.
10157 }

10158 \@@_msg_new:nn { cellcolor-in-Block }
10159 {
10160     Bad-use-of~\token_to_str:N \cellcolor \\
10161     You-can't-use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10162     \bool_if:NTF \l_@@_amp_in_blocks_bool
10163     { (but-you-could-use-it-in-a-sub-block~since~'&-in-blocks'~is~in-force) }

```

```

10164     { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10165     .~Here,~you~should~use~the~key~'fill'~of~the~block.\\
10166     That~command~will~be~ignored.
10167 }
10168 \@@_msg_new:nn { rowcolor~in~Block }
10169 {
10170     Bad~use~of~\token_to_str:N \rowcolor \\
10171     You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10172     That~command~will~be~ignored.
10173 }
10174 \@@_msg_new:nn { key~color~inside }
10175 {
10176     Deleted~key.\\
10177     The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10178     ~'nicematrix'~and~must~not~be~used.\\
10179     This~error~is~fatal.
10180 }
10181 \@@_msg_new:nn { invalid~weight }
10182 {
10183     Unknown~key.\\
10184     The~key~'\l_keys_key_str'~of~your~column~X~is~unknown~and~will~be~ignored.~
10185     The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10186     \IfPackageLoadedTF { varwidth }
10187     { (since~'varwidth'~is~loaded)~}
10188     { (if~you~load~'varwidth')~}
10189     and~real~numbers~for~the~weight~of~the~X~column.
10190 }
10191 \@@_msg_new:nn { last~col~not~used }
10192 {
10193     Column~not~used.\\
10194     The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
10195     in~your~\@@_full_name_env: .~
10196     However,~you~can~go~on.
10197 }
10198 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10199 {
10200     Too~many~columns.\\
10201     In~the~row~\int_eval:n { \c@iRow },~
10202     you~try~to~use~more~columns~
10203     than~allowed~by~your~\@@_full_name_env: .
10204     \@@_message_hdotsfor: \
10205     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
10206     (plus~the~exterior~columns).~This~error~is~fatal.
10207 }
10208 \@@_msg_new:nn { too~many~cols~for~matrix }
10209 {
10210     Too~many~columns.\\
10211     In~the~row~\int_eval:n { \c@iRow },~
10212     you~try~to~use~more~columns~than~allowed~by~your~\@@_full_name_env: .
10213     \@@_message_hdotsfor: \
10214     Recall~that~the~maximal~number~of~columns~for~a~matrix~
10215     (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10216     LaTeX~counter~'MaxMatrixCols'.~
10217     Its~current~value~is~\int_use:N \c@MaxMatrixCols \
10218     (use~\token_to_str:N \setcounter \ to~change~that~value).~
10219     This~error~is~fatal.
10220 }
10221 \@@_msg_new:nn { too~many~cols~for~array }
10222 {
10223     Too~many~columns.\\

```

```

10224 In~the~row~ \int_eval:n { \c@iRow } ,~
10225 ~you~try~to~use~more~columns~than~allowed~by~your~
10226 \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10227 \int_use:N \g_@@_static_num_of_col_int \
10228 \bool_if:nT
10229 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10230 { (plus~the~exterior~ones)~}
10231 since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
10232 This~error~is~fatal.
10233 }
10234 \@@_msg_new:nn { columns~not~used }
10235 {
10236 Columns~not~used.\\
10237 The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10238 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10239 columns~but~you~only~used~ \int_use:N \c@jCol .\\
10240 The~columns~you~did~not~used~won't~be~created.\\
10241 You~won't~have~similar~warning~till~the~end~of~the~document.
10242 }
10243 \@@_msg_new:nn { empty~preamble }
10244 {
10245 Empty~preamble.\\
10246 The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10247 This~error~is~fatal.
10248 }
10249 \@@_msg_new:nn { in~first~col }
10250 {
10251 Erroneous~use.\\
10252 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
10253 That~command~will~be~ignored.\\
10254 You~can~try~to~delete~the~key~'first~col'.
10255 }
10256 \@@_msg_new:nn { in~last~col }
10257 {
10258 Erroneous~use.\\
10259 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
10260 That~command~will~be~ignored.\\
10261 You~can~try~to~delete~the~key~'last~col'.
10262 }
10263 \@@_msg_new:nn { in~first~row }
10264 {
10265 Erroneous~use.\\
10266 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10267 That~command~will~be~ignored.\\
10268 You~can~try~to~delete~the~key~'first~row'.
10269 }
10270 \@@_msg_new:nn { in~last~row }
10271 {
10272 Erroneous~use.\\
10273 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10274 That~command~will~be~ignored.\\
10275 You~can~try~to~delete~the~key~'last~row'.
10276 }
10277 \@@_msg_new:nn { TopRule~without~booktabs }
10278 {
10279 Erroneous~use.\\
10280 You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10281 That~command~will~be~ignored.
10282 }
10283 \@@_msg_new:nn{ rotate~in~p~col }

```

```

10284 {
10285   \token_to_str:N \rotate\ forbidden.\\
10286   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10287   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10288   If~you~go~on,~maybe~you~won't~have~the~expected~output.
10289 }
10290 \@@_msg_new:nn { TopRule~without~tikz }
10291 {
10292   Erroneous~use.\\
10293   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.\\
10294   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
10295   \IfPackageLoadedF { booktabs }
10296   { You~should~also~load~'booktabs'.\\ }
10297   That~command~will~be~ignored.
10298 }
10299 \@@_msg_new:nn { caption~outside~float }
10300 {
10301   Key~caption~forbidden.\\
10302   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10303   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10304 }
10305 \@@_msg_new:nn { short~caption~without~caption }
10306 {
10307   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10308   However,~your~'short~caption'~will~be~used~as~'caption'.
10309 }
10310 \@@_msg_new:nn { double~closing~delimiter }
10311 {
10312   Double~delimiter.\\
10313   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10314   delimiter.~This~delimiter~will~be~ignored.\\
10315   You~can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10316 }
10317 \@@_msg_new:nn { delimiter~after~opening }
10318 {
10319   Double~delimiter.\\
10320   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10321   delimiter.~That~delimiter~will~be~ignored.
10322 }
10323 \@@_msg_new:nn { bad~option~for~line~style }
10324 {
10325   Bad~line~style.\\
10326   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10327   is~'standard'.~That~key~will~be~ignored.\\
10328   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
10329 }
10330 \@@_msg_new:nn { corners~with~no~cell~nodes }
10331 {
10332   Incompatible~keys.\\
10333   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10334   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10335   is~to~speed~up~compilation).\\
10336   If~you~go~on,~that~key~will~be~ignored.
10337 }
10338 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10339 {
10340   Incompatible~keys.\\
10341   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10342   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10343   is~to~speed~up~compilation).\\

```

```

10344     If~you~go~on,~those~extra~nodes~won't~be~created.
10345 }

10346 \@@_msg_new:nn { Identical~notes~in~caption }
10347 {
10348     Identical~tabular~notes.\\
10349     You~can't~put~several~notes~with~the~same~content~in~
10350     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\\
10351     If~you~go~on,~the~output~will~probably~be~erroneous.
10352 }

10353 \@@_msg_new:nn { tabularnote~below~the~tabular }
10354 {
10355     \token_to_str:N \tabularnote \ forbidden\\
10356     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10357     of~your~tabular~because~the~caption~will~be~composed~below~
10358     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10359     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10360     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10361     no~similar~error~will~raised~in~this~document.
10362 }

10363 \@@_msg_new:nn { Unknown~key~for~rules }
10364 {
10365     Unknown~key.\\
10366     There~is~only~two~keys~available~here:~width~and~color.\\
10367     Your~key~' \l_keys_key_str '~will~be~ignored.
10368 }

10369 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10370 {
10371     Unknown~key.\\
10372     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10373     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10374     and~ \token_to_str:N \Vbrace \ are:~'brace~shift(+)',~'color',~
10375     'horizontal~label(s)',~'shorten'~'shorten~end'~
10376     and~'shorten~start'.\\
10377     That~error~is~fatal.
10378 }

10379 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10380 {
10381     Unknown~key.\\
10382     There~is~only~two~keys~available~here:~
10383     'empty'~and~'not~empty'.\\
10384     Your~key~' \l_keys_key_str '~will~be~ignored.
10385 }

10386 \@@_msg_new:nn { Unknown~key~for~rotate }
10387 {
10388     Unknown~key.\\
10389     The~only~key~available~here~is~'c'.\\
10390     Your~key~' \l_keys_key_str '~will~be~ignored.
10391 }

10392 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10393 {
10394     Unknown~key.\\
10395     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10396     It~you~go~on,~you~will~probably~have~other~errors. \\
10397     \c_@@_available_keys_str
10398 }
10399 {
10400     The~available~keys~are~(in~alphabetic~order):~
10401     ccommand,~
10402     color,~
10403     command,~
10404     dotted,~

```

```

10405     letter,~
10406     multiplicity,~
10407     sep-color,~
10408     tikz,~and~total-width.
10409 }
10410 \@@_msg_new:nnn { Unknown~key~for~xdots }
10411 {
10412     Unknown~key.\\
10413     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10414     \c_@@_available_keys_str
10415 }
10416 {
10417     The~available~keys~are~(in~alphabetic~order):~
10418     'color',~
10419     'horizontal(s)-labels',~
10420     'inter',~
10421     'line-style',~
10422     'nullify',~
10423     'radius',~
10424     'shorten',~
10425     'shorten-end'~and~'shorten-start'.
10426 }
10427 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10428 {
10429     Unknown~key.\\
10430     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10431     (and~you~try~to~use~' \l_keys_key_str '~)\\
10432     That~key~will~be~ignored.
10433 }
10434 \@@_msg_new:nn { label~without~caption }
10435 {
10436     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10437     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10438 }
10439 \@@_msg_new:nn { W~warning }
10440 {
10441     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10442     (row~ \int_use:N \c@iRow ).
10443 }
10444 \@@_msg_new:nn { Construct~too~large }
10445 {
10446     Construct~too~large.\\
10447     Your~command~ \token_to_str:N #1
10448     can't~be~drawn~because~your~matrix~is~too~small.\\
10449     That~command~will~be~ignored.
10450 }
10451 \@@_msg_new:nn { underscore~after~nicematrix }
10452 {
10453     Problem~with~'underscore'.\\
10454     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10455     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10456     ' \token_to_str:N \Cdots \token_to_str:N _
10457     \{ n \token_to_str:N \text \{ ~times \} \}'.
10458 }
10459 \@@_msg_new:nn { ampersand~in~light-syntax }
10460 {
10461     Ampersand~forbidden.\\
10462     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10463     the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10464 }

```

```

10465 \@@_msg_new:nn { double-backslash~in~light-syntax }
10466 {
10467   Double~backslash~forbidden.\\
10468   You~can't~use~ \token_to_str:N \\
10469   ~to~separate~rows~because~the~key~'light-syntax'~
10470   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10471   (set~by~the~key~'end-of-row').~This~error~is~fatal.
10472 }
10473 \@@_msg_new:nn { hlines~with~color }
10474 {
10475   Incompatible~keys.\\
10476   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10477   \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10478   However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10479   Your~key~will~be~discarded.
10480 }
10481 \@@_msg_new:nn { bad~value~for~baseline }
10482 {
10483   Bad~value~for~baseline.\\
10484   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10485   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10486   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10487   the~form~'line-i'.\\
10488   A~value~of~1~will~be~used.
10489 }
10490 \@@_msg_new:nn { bad~value~for~baseline~line }
10491 {
10492   Bad~value~for~baseline~with~line.\\
10493   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10494   valid.~The~number~of~the~line~must~be~between~1~and~
10495   \int_eval:n { \c@iRow + 1 } \\
10496   A~value~of~'line-1'~will~be~used.
10497 }
10498 \@@_msg_new:nn { detection~of~empty~cells }
10499 {
10500   Problem~with~'not~empty'\\
10501   For~technical~reasons,~you~must~activate~
10502   'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
10503   in~order~to~use~the~key~' \l_keys_key_str '.\\
10504   That~key~will~be~ignored.
10505 }
10506 \@@_msg_new:nn { siunitx~not~loaded }
10507 {
10508   siunitx~not~loaded\\
10509   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
10510   That~error~is~fatal.\\
10511   You~can~load~'siunitx'~with~\token_to_str:N \usepackage \{siunitx\}.\\
10512 }
10513 \@@_msg_new:nn { Invalid~name }
10514 {
10515   Invalid~name.\\
10516   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10517   \SubMatrix \ of~your~ \@@_full_name_env: .\\
10518   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
10519   This~key~will~be~ignored.
10520 }
10521 \@@_msg_new:nn { Hbrace~not~allowed }
10522 {
10523   Command~not~allowed.\\
10524   You~can't~use~the~command~ \token_to_str:N #1
10525   because~you~have~not~loaded~

```

```

10526 \IfPackageLoadedTF { tikz }
10527 { theTikZ-library~'decorations.pathreplacing'~Use~ }
10528 { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10529 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10530 That~command~will~be~ignored.
10531 }
10532 \@@_msg_new:nn { Vbrace~not~allowed }
10533 {
10534 Command~not~allowed.\\
10535 You~can't~use~the~command~ \token_to_str:N \Vbrace \
10536 because~you~have~not~loaded~TikZ~
10537 and~the~TikZ-library~'decorations.pathreplacing'\\.\\
10538 Use: ~\token_to_str:N \usepackage \{tikz\}~
10539 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10540 That~command~will~be~ignored.
10541 }
10542 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10543 {
10544 Wrong~line.\\
10545 You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10546 \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10547 number~is~not~valid.~It~will~be~ignored.
10548 }
10549 \@@_msg_new:nn { Impossible~delimiter }
10550 {
10551 Impossible~delimiter.\\
10552 It's~impossible~to~draw~the~#1~delimiter~of~your~
10553 \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10554 in~that~column.
10555 \bool_if:NT \l_@@_submatrix_slim_bool
10556 { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10557 This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10558 }
10559 \@@_msg_new:nnn { width~without~X~columns }
10560 {
10561 You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10562 the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10563 That~key~will~be~ignored.
10564 }
10565 {
10566 This~message~is~the~message~'width~without~X~columns'~
10567 of~the~module~'nicematrix'.~
10568 The~experimented~users~can~disable~that~message~with~
10569 \token_to_str:N \msg_redirect_name:nnn .\\
10570 }
10571
10572 \@@_msg_new:nn { key~multiplicity~with~dotted }
10573 {
10574 Incompatible~keys. \\
10575 You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10576 in~a~'custom~line'~.~They~are~incompatible. \\
10577 The~key~'multiplicity'~will~be~discarded.
10578 }
10579 \@@_msg_new:nn { empty~environment }
10580 {
10581 Empty~environment.\\
10582 Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10583 }
10584 \@@_msg_new:nn { No~letter~and~no~command }
10585 {
10586 Erroneous~use.\\

```



```

10587     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
10588     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10589     ~'ccommand'~(to~draw~horizontal~rules).\\
10590     However,~you~can~go~on.
10591 }
10592 \@@_msg_new:nn { Forbidden~letter }
10593 {
10594     Forbidden~letter.\\
10595     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10596     It~will~be~ignored.\\
10597     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10598 }
10599 \@@_msg_new:nn { Several~letters }
10600 {
10601     Wrong~name.\\
10602     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10603     have~used~' \l_@@_letter_str ').\\
10604     It~will~be~ignored.
10605 }
10606 \@@_msg_new:nn { Delimiter~with~small }
10607 {
10608     Delimiter~forbidden.\\
10609     You~can't~put~a~delimiter~in~the~preamble~of~your~
10610     \@@_full_name_env: \
10611     because~the~key~'small'~is~in~force.\\
10612     This~error~is~fatal.
10613 }
10614 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10615 {
10616     Unknown~cell.\\
10617     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10618     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10619     can't~be~executed~because~a~cell~doesn't~exist.\\
10620     This~command~ \token_to_str:N \line \ will~be~ignored.
10621 }
10622 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10623 {
10624     Duplicate~name.\\
10625     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10626     in~this~ \@@_full_name_env: .\\
10627     This~key~will~be~ignored.\\
10628     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10629     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10630 }
10631 {
10632     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10633     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10634 }
10635 \@@_msg_new:nn { r~or~l~with~preamble }
10636 {
10637     Erroneous~use.\\
10638     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10639     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10640     your~ \@@_full_name_env: .\\
10641     This~key~will~be~ignored.
10642 }
10643 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10644 {
10645     Erroneous~use.\\
10646     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10647     in~an~exterior~column~of~

```

```

10648     the~array.~This~error~is~fatal.
10649 }

10650 \@@_msg_new:nn { bad~corner }
10651 {
10652     Bad~corner.\\
10653     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10654     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10655     This~specification~of~corner~will~be~ignored.
10656 }

10657 \@@_msg_new:nn { bad~border }
10658 {
10659     Bad~border.\\
10660     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10661     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10662     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10663     also~use~the~key~'tikz'
10664     \IfPackageLoadedF { tikz }
10665     { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10666     This~specification~of~border~will~be~ignored.
10667 }

10668 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10669 {
10670     TikZ~not~loaded.\\
10671     You~can't~use~ \token_to_str:N \TikzEveryCell \
10672     because~you~have~not~loaded~tikz.\\
10673     You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\}.\\
10674     This~command~will~be~ignored.
10675 }

10676 \@@_msg_new:nn { tikz~key~without~tikz }
10677 {
10678     TikZ~not~loaded.\\
10679     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10680     \Block '~because~you~have~not~loaded~tikz.\\
10681     You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\}.\\
10682     This~key~will~be~ignored.
10683 }

10684 \@@_msg_new:nn { Bad~argument~for~Block }
10685 {
10686     Bad~argument.\\
10687     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10688     be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
10689     '#1'. \\
10690     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
10691     the~argument~was~empty).
10692 }

10693 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10694 {
10695     Erroneous~use.\\
10696     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10697     'last~col'~without~value.\\
10698     However,~you~can~go~on~for~this~time~
10699     (the~value~' \l_keys_value_tl '~will~be~ignored).
10700 }

10701 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10702 {
10703     Erroneous~use. \\
10704     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10705     'last~col'~without~value. \\
10706     However,~you~can~go~on~for~this~time~
10707     (the~value~' \l_keys_value_tl '~will~be~ignored).
10708 }

```

```

10709 \@@_msg_new:nn { Block-too-large-1 }
10710 {
10711   Block-too-large. \\
10712   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is-
10713   too-small-for-that-block. \\
10714   This-block-and-maybe-others-will-be-ignored.
10715 }
10716 \@@_msg_new:nn { Block-too-large-2 }
10717 {
10718   Block-too-large. \\
10719   The-preamble-of-your- \@@_full_name_env: \ announces- \int_use:N
10720   \g_@@_static_num_of_col_int \
10721   columns-but-you-use-only- \int_use:N \c@jCol \ and-that's-why-a-block-
10722   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands-
10723   (&)~at-the-end-of-the-first-row-of-your- \@@_full_name_env: . \\
10724   This-block-and-maybe-others-will-be-ignored.
10725 }
10726 \@@_msg_new:nn { unknown-column-type }
10727 {
10728   Bad-column-type. \\
10729   The-column-type-#1'-in-your- \@@_full_name_env: \
10730   is-unknown. \\
10731   This-error-is-fatal.
10732 }
10733 \@@_msg_new:nn { unknown-column-type-multicolumn }
10734 {
10735   Bad-column-type. \\
10736   The-column-type-#1'-in-the-command-\token_to_str:N \multicolumn \
10737   ~of-your- \@@_full_name_env: \
10738   is-unknown. \\
10739   This-error-is-fatal.
10740 }
10741 \@@_msg_new:nn { unknown-column-type-S }
10742 {
10743   Bad-column-type. \\
10744   The-column-type-'S'-in-your- \@@_full_name_env: \ is-unknown. \\
10745   If-you-want-to-use-the-column-type-'S'-of-siunitx,~you-should-
10746   load-that-package. \\
10747   This-error-is-fatal.
10748 }
10749 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
10750 {
10751   Bad-column-type. \\
10752   The-column-type-'S'-in-the-command-\token_to_str:N \multicolumn \
10753   of-your- \@@_full_name_env: \ is-unknown. \\
10754   If-you-want-to-use-the-column-type-'S'-of-siunitx,~you-should-
10755   load-that-package. \\
10756   This-error-is-fatal.
10757 }
10758 \@@_msg_new:nn { tabularnote-forbidden }
10759 {
10760   Forbidden-command. \\
10761   You-can't-use-the-command- \token_to_str:N \tabularnote \
10762   ~here.~This-command-is-available-only-in-
10763   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in-
10764   the-argument-of-a-command-\token_to_str:N \caption \ included-
10765   in-an-environment-\{table\}. \\
10766   This-command-will-be-ignored.
10767 }
10768 \@@_msg_new:nn { borders-forbidden }
10769 {

```

```

10770     Forbidden-key.\\
10771     You-can't-use-the-key~'borders'~of~the~command~ \token_to_str:N \Block \
10772     because~the~option~'rounded-corners'~
10773     is~in~force~with~a~non-zero-value.\\
10774     This-key~will~be~ignored.
10775 }
10776 \@@_msg_new:nn { bottomrule~without~booktabs }
10777 {
10778     booktabs~not~loaded.\\
10779     You-can't-use-the-key~'tabular/bottomrule'~because~you~haven't~
10780     loaded~'booktabs'.\\
10781     This-key~will~be~ignored.
10782 }
10783 \@@_msg_new:nn { enumitem~not~loaded }
10784 {
10785     enumitem~not~loaded. \\
10786     You-can't-use-the-command~ \token_to_str:N \tabularnote \
10787     ~because~you~haven't~loaded~'enumitem'. \\
10788     All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10789     ignored~in~the~document.
10790 }
10791 \@@_msg_new:nn { tikz~without~tikz }
10792 {
10793     TikZ~not~loaded. \\
10794     You-can't-use-the-key~'tikz'~here~because~TikZ~is~not~
10795     loaded.~If~you~go~on,~that~key~will~be~ignored.
10796 }
10797 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10798 {
10799     TikZ~not~loaded. \\
10800     You~have~used~the~key~'tikz'~in~the~definition~of~a~
10801     customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
10802     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10803     use~that~custom~line.
10804 }
10805 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10806 {
10807     TikZ~not~loaded. \\
10808     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10809     command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
10810     That~key~will~be~ignored.
10811 }
10812 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10813 {
10814     Erroneous~use.\\
10815     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10816     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10817     The~key~'color'~will~be~discarded.
10818 }
10819 \@@_msg_new:nn { Wrong~last~row }
10820 {
10821     Wrong~number.\\
10822     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10823     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10824     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10825     last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10826     problem~by~using~'last~row'~without~value~(more~compilations~
10827     might~be~necessary).
10828 }
10829 \@@_msg_new:nn { Yet~in~env }

```

```

10830 {
10831   Nested~environments.\\
10832   Environments-of~nicematrix~can't-be-nested.\\
10833   This~error~is~fatal.
10834 }
10835 \@@_msg_new:nn { Outside~math~mode }
10836 {
10837   Outside~math~mode.\\
10838   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10839   (and~not~in~ \token_to_str:N \vcenter ).\\
10840   This~error~is~fatal.
10841 }
10842 \@@_msg_new:nn { One~letter~allowed }
10843 {
10844   Bad~name.\\
10845   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10846   you~have~used~' \l_keys_value_tl '~.\\
10847   It~will~be~ignored.
10848 }
10849 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10850 {
10851   Environment~\{TabularNote\}~forbidden.\\
10852   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10853   but~*before*~the~ \token_to_str:N \CodeAfter . \\
10854   This~environment~\{TabularNote\}~will~be~ignored.
10855 }
10856 \@@_msg_new:nn { varwidth~not~loaded }
10857 {
10858   varwidth~not~loaded.\\
10859   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10860   loaded.\\
10861   Your~column~will~behave~like~'p'.
10862 }
10863 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10864 {
10865   varwidth~not~loaded.\\
10866   You~can't~use~the~key~'V'~in~your~column~'X'~
10867   because~'varwidth'~is~not~loaded.\\
10868   It~will~be~ignored. \\
10869 }
10870 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10871 {
10872   Unknown~key.\\
10873   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10874   \c_@@_available_keys_str
10875 }
10876 {
10877   The~available~keys~are~(in~alphabetic~order):~
10878   color,~
10879   dotted,~
10880   multiplicity,~
10881   sep~color,~
10882   tikz,~and~total~width.
10883 }
10884
10885 \@@_msg_new:nnn { Unknown~key~for~Block }
10886 {
10887   Unknown~key. \\
10888   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10889   \token_to_str:N \Block . \\
10890   It~will~be~ignored. \\

```

```

10891 \c_@@_available_keys_str
10892 }
10893 {
10894   The-available-keys-are-(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
10895   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10896   opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10897   and~vlines.
10898 }
10899 \@@_msg_new:nnn { Unknown~key~for~Brace }
10900 {
10901   Unknown~key.\\
10902   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10903   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10904   It~will~be~ignored. \\
10905   \c_@@_available_keys_str
10906 }
10907 {
10908   The-available-keys-are-(in~alphabetic~order):~color,~left-shorten,~
10909   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10910   right-shorten)~and~yshift.
10911 }
10912 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10913 {
10914   Unknown~key.\\
10915   The~key~' \l_keys_key_str '~is~unknown.\\
10916   It~will~be~ignored. \\
10917   \c_@@_available_keys_str
10918 }
10919 {
10920   The-available-keys-are-(in~alphabetic~order):~
10921   delimiters/color,~
10922   rules~(with~the~subkeys~'color'~and~'width'),~
10923   sub-matrix~(several~subkeys)~
10924   and~xdots~(several~subkeys).~
10925   The~latter~is~for~the~command~ \token_to_str:N \line .
10926 }
10927 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10928 {
10929   Unknown~key.\\
10930   The~key~' \l_keys_key_str '~is~unknown.\\
10931   It~will~be~ignored. \\
10932   \c_@@_available_keys_str
10933 }
10934 {
10935   The-available-keys-are-(in~alphabetic~order):~
10936   create-cell-nodes,~
10937   delimiters/color~and~
10938   sub-matrix~(several~subkeys).
10939 }
10940 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10941 {
10942   Unknown~key.\\
10943   The~key~' \l_keys_key_str '~is~unknown.\\
10944   That~key~will~be~ignored. \\
10945   \c_@@_available_keys_str
10946 }
10947 {
10948   The-available-keys-are-(in~alphabetic~order):~
10949   'delimiters/color',~
10950   'extra-height',~
10951   'hlines',~
10952   'hvlines',~

```

```

10953     'left-xshift',~
10954     'name',~
10955     'right-xshift',~
10956     'rules'~(with-the-subkeys~'color'~and~'width'),~
10957     'slim',~
10958     'vlines'~and~'xshift'~(which-sets-both~'left-xshift'~
10959     and~'right-xshift').\\
10960 }
10961 \@@_msg_new:nnn { Unknown~key~for~notes }
10962 {
10963     Unknown~key.\\
10964     The~key~' \l_keys_key_str '~is~unknown.\\
10965     That~key~will~be~ignored. \\
10966     \c_@@_available_keys_str
10967 }
10968 {
10969     The~available~keys~are~(in~alphabetic~order):~
10970     bottomrule,~
10971     code~after,~
10972     code~before(+),~
10973     detect~duplicates,~
10974     enumitem~keys,~
10975     enumitem~keys~para,~
10976     para,~
10977     label~in~list,~
10978     label~in~tabular~and~
10979     style.
10980 }
10981 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10982 {
10983     Unknown~key.\\
10984     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10985     \token_to_str:N \RowStyle . \\
10986     That~key~will~be~ignored. \\
10987     \c_@@_available_keys_str
10988 }
10989 {
10990     The~available~keys~are~(in~alphabetic~order):~
10991     bold,~
10992     cell~space~top~limit(+),~
10993     cell~space~bottom~limit(+),~
10994     cell~space~limits(+),~
10995     color,~
10996     fill~(alias:~rowcolor),~
10997     nb~rows,~
10998     opacity~and~
10999     rounded~corners.
11000 }
11001 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11002 {
11003     Unknown~key.\\
11004     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11005     \token_to_str:N \NiceMatrixOptions . \\
11006     That~key~will~be~ignored. \\
11007     \c_@@_available_keys_str
11008 }
11009 {
11010     The~available~keys~are~(in~alphabetic~order):~
11011     &~in~blocks,~
11012     allow~duplicate~names,~
11013     ampersand~in~blocks,~
11014     caption~above,~
11015     cell~space~bottom~limit(+),~

```

```

11016 cell-space-limits(+),~
11017 cell-space-top-limit(+),~
11018 code-for-first-col(+),~
11019 code-for-first-row(+),~
11020 code-for-last-col(+),~
11021 code-for-last-row(+),~
11022 corners,~
11023 custom-key,~
11024 create-extra-nodes,~
11025 create-medium-nodes,~
11026 create-large-nodes,~
11027 custom-line,~
11028 delimiters~(several~subkeys),~
11029 end-of-row,~
11030 first-col,~
11031 first-row,~
11032 hlines,~
11033 hvlines,~
11034 hvlines-except-borders,~
11035 last-col,~
11036 last-row,~
11037 left-margin,~
11038 light-syntax,~
11039 light-syntax-expanded,~
11040 matrix/columns-type,~
11041 no-cell-nodes,~
11042 notes~(several~subkeys),~
11043 nullify-dots,~
11044 pgf-node-code,~
11045 renew-dots,~
11046 renew-matrix,~
11047 respect-arraystretch,~
11048 rounded-corners,~
11049 right-margin,~
11050 rules~(with~the~subkeys~'color'~and~'width'),~
11051 small,~
11052 sub-matrix~(several~subkeys),~
11053 vlines,~
11054 xdots~(several~subkeys).
11055 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11056 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11057 {
11058   Unknown~key.\\
11059   The~key~' \l_keys_key_str '-is~unknown~for~the~environment~
11060   \{NiceArray\}. \\
11061   That~key~will~be~ignored. \\
11062   \c_@@_available_keys_str
11063 }
11064 {
11065   The~available~keys~are~(in~alphabetic~order):~
11066   &~in~blocks,~
11067   ampersand~in~blocks,~
11068   b,~
11069   baseline,~
11070   c,~
11071   cell-space-bottom-limit,~
11072   cell-space-limits,~
11073   cell-space-top-limit,~
11074   code~after,~
11075   code~for~first~col(+),~
11076   code~for~first~row(+),~

```



```

11077 code-for-last-col(+),~
11078 code-for-last-row(+),~
11079 columns-width,~
11080 corners,~
11081 create-blocks-in-col,~
11082 create-extra-nodes,~
11083 create-medium-nodes,~
11084 create-large-nodes,~
11085 extra-left-margin,~
11086 extra-right-margin,~
11087 first-col,~
11088 first-row,~
11089 hlines,~
11090 hvlines,~
11091 hvlines-except-borders,~
11092 last-col,~
11093 last-row,~
11094 left-margin,~
11095 light-syntax,~
11096 light-syntax-expanded,~
11097 name,~
11098 no-cell-nodes,~
11099 nullify-dots,~
11100 pgf-node-code,~
11101 renew-dots,~
11102 respect-arraystretch,~
11103 right-margin,~
11104 rounded-corners,~
11105 rules~(with~the~subkeys~'color'~and~'width'),~
11106 small,~
11107 t,~
11108 vlines,~
11109 xdots/color,~
11110 xdots/shorten-start(+),~
11111 xdots/shorten-end(+),~
11112 xdots/shorten(+),~and~
11113 xdots/line-style.
11114 }

```

This error message is used for the set of keys `nicematrix/NiceMatrix` and `nicematrix/pNiceArray` (but not by `nicematrix/NiceArray` because, for this set of keys, there is no `l` and `r`).

```

11115 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11116 {
11117   Unknown~key.\\
11118   The~key~' \l_keys_key_str '~is~unknown~for~the~
11119   \@@_full_name_env: . \\
11120   That~key~will~be~ignored. \\
11121   \c_@@_available_keys_str
11122 }
11123 {
11124   The~available~keys~are~(in~alphabetic~order):~
11125   &~in~blocks,~
11126   ampersand~in~blocks,~
11127   b,~
11128   baseline,~
11129   c,~
11130   cell-space-bottom-limit,~
11131   cell-space-limits,~
11132   cell-space-top-limit,~
11133   code~after,~
11134   code~for~first~col(+),~
11135   code~for~first~row(+),~
11136   code~for~last~col(+),~

```

```

11137 code-for-last-row(+),~
11138 columns-type,~
11139 columns-width,~
11140 corners,~
11141 create-blocks-in-col,~
11142 create-extra-nodes,~
11143 create-medium-nodes,~
11144 create-large-nodes,~
11145 extra-left-margin,~
11146 extra-right-margin,~
11147 first-col,~
11148 first-row,~
11149 hlines,~
11150 hvlines,~
11151 hvlines-except-borders,~
11152 l,~
11153 last-col,~
11154 last-row,~
11155 left-margin,~
11156 light-syntax,~
11157 light-syntax-expanded,~
11158 name,~
11159 no-cell-nodes,~
11160 nullify-dots,~
11161 pgf-node-code,~
11162 r,~
11163 renew-dots,~
11164 respect-arraystretch,~
11165 right-margin,~
11166 rounded-corners,~
11167 rules~(with~the~subkeys~'color'~and~'width'),~
11168 small,~
11169 t,~
11170 vlines,~
11171 xdots/color,~
11172 xdots/shorten-start(+),~
11173 xdots/shorten-end(+),~
11174 xdots/shorten(+)-and~
11175 xdots/line-style.
11176 }
11177 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11178 {
11179   Unknown~key.\\
11180   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11181   \{NiceTabular\}. \\
11182   That~key~will~be~ignored. \\
11183   \c_@@_available_keys_str
11184 }
11185 {
11186   The~available~keys~are~(in~alphabetic~order):~
11187   &-in-blocks,~
11188   ampersand-in-blocks,~
11189   b,~
11190   baseline,~
11191   c,~
11192   caption,~
11193   cell-space-bottom-limit,~
11194   cell-space-limits,~
11195   cell-space-top-limit,~
11196   code-after,~
11197   code-for-first-col(+),~
11198   code-for-first-row(+),~
11199   code-for-last-col(+),~

```

```

11200 code-for-last-row(+),~
11201 columns-width,~
11202 corners,~
11203 custom-line,~
11204 create-blocks-in-col,~
11205 create-extra-nodes,~
11206 create-medium-nodes,~
11207 create-large-nodes,~
11208 extra-left-margin,~
11209 extra-right-margin,~
11210 first-col,~
11211 first-row,~
11212 hlines,~
11213 hvlines,~
11214 hvlines-except-borders,~
11215 label,~
11216 last-col,~
11217 last-row,~
11218 left-margin,~
11219 light-syntax,~
11220 light-syntax-expanded,~
11221 name,~
11222 no-cell-nodes,~
11223 notes~(several~subkeys),~
11224 nullify-dots,~
11225 pgf-node-code,~
11226 renew-dots,~
11227 respect-arraystretch,~
11228 right-margin,~
11229 rounded-corners,~
11230 rules~(with~the~subkeys~'color'~and~'width'),~
11231 short-caption,~
11232 t,~
11233 tabularnote,~
11234 vlides,~
11235 xdots/color,~
11236 xdots/shorten-start(+),~
11237 xdots/shorten-end(+),~
11238 xdots/shorten(+)-and~
11239 xdots/line-style.
11240 }
11241 \@@_msg_new:nnn { Duplicate~name }
11242 {
11243   Duplicate~name.\\
11244   The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
11245   the~same~environment~name~twice.~You~can~go~on,~but,~
11246   maybe,~you~will~have~incorrect~results~especially~
11247   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11248   message~again,~use~the~key~'allow-duplicate-names'~in~
11249   ' \token_to_str:N \NiceMatrixOptions '~\\
11250   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11251   { For~a~list~of~the~names~already~used,~type~H~<return>. }
11252 }
11253 {
11254   The~names~already~defined~in~this~document~are::~
11255   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11256 }
11257 \@@_msg_new:nn { caption-above~in~env }
11258 {
11259   The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
11260   That~key~will~be~ignored.
11261 }
11262 \@@_msg_new:nn { show-cell-names }

```

```

11263 {
11264     There-is-no-key~'show-cell-names'~in~nicematrix.\\
11265     You-should-use-the-command~\token_to_str:N \ShowCellNames\
11266     in-the~\token_to_str:N \CodeBefore\ or-the~\token_to_str:N
11267     \CodeAfter. \\
11268     That-key-will-be-ignored.
11269 }
11270 \@@_msg_new:nn { Option~auto~for~columns-width }
11271 {
11272     Erroneous-use.\\
11273     You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
11274     That-key-will-be-ignored.
11275 }
11276 \@@_msg_new:nn { NiceTabularX~without~X }
11277 {
11278     NiceTabularX~without~X.\\
11279     You-should-not-use~\{NiceTabularX\}~without~X~columns.\\
11280     However,~you~can~go~on.
11281 }
11282 \@@_msg_new:nn { Preamble~forgotten }
11283 {
11284     Preamble-forgotten.\\
11285     You-have-probably-forgotten-the-preamble-of-your~
11286     \@@_full_name_env: . \\
11287     This-error-is-fatal.
11288 }
11289 \@@_msg_new:nn { Invalid~col~number }
11290 {
11291     Invalid~column~number.\\
11292     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11293     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
11294 }
11295 \@@_msg_new:nn { Invalid~row~number }
11296 {
11297     Invalid~row~number.\\
11298     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11299     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
11300 }
11301 \@@_define_com:NNN p ( )
11302 \@@_define_com:NNN b [ ]
11303 \@@_define_com:NNN v | |
11304 \@@_define_com:NNN V \l \l
11305 \@@_define_com:NNN B \{ \}

```

# Contents

|    |  |     |
|----|--|-----|
| 1  | Declaration of the package and packages loaded   | 1   |
| 2  | Collecting options   | 3   |
| 3  | Technical definitions  | 3   |
| 4  | Parameters   | 9   |
| 5  | The command <code>\tabularnote</code>  | 20  |
| 6  | Command for creation of rectangle nodes  | 25  |
| 7  | The options  | 26  |
| 8  | Important code used by <code>{NiceArrayWithDelims}</code>                                | 38  |
| 9  | The <code>\CodeBefore</code>   | 54  |
| 10 | The environment <code>{NiceArrayWithDelims}</code>                                       | 58  |
| 11 | Construction of the preamble of the array  | 63  |
| 12 | The redefinition of <code>\multicolumn</code>  | 80  |
| 13 | The environment <code>{NiceMatrix}</code> and its variants                               | 97  |
| 14 | <code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code> | 98  |
| 15 | After the construction of the array  | 100 |
| 16 | We draw the dotted lines   | 106 |
| 17 | The actual instructions for drawing the dotted lines with <code>TikZ</code>              | 124 |
| 18 | User commands available in the new environments  | 130 |
| 19 | The command <code>\line</code> accessible in code-after                                  | 136 |
| 20 | The command <code>\RowStyle</code>   | 138 |
| 21 | Colors of cells, rows and columns  | 141 |
| 22 | The vertical and horizontal rules  | 153 |
| 23 | The empty corners  | 170 |
| 24 | The environment <code>{NiceMatrixBlock}</code>   | 172 |
| 25 | The extra nodes  | 174 |
| 26 | The blocks   | 178 |
| 27 | How to draw the dotted lines transparently   | 205 |
| 28 | Automatic arrays   | 205 |
| 29 | The redefinition of the command <code>\dotfill</code>                                    | 207 |
| 30 | The command <code>\diagbox</code>  | 207 |

|    |   |     |
|----|---|-----|
| 31 | The keyword <code>\CodeAfter</code>                               | 208 |
| 32 | The delimiters in the preamble                                    | 209 |
| 33 | The command <code>\SubMatrix</code>                               | 210 |
| 34 | Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code> | 219 |
| 35 | The commands <code>HBrace</code> et <code>VBrace</code>           | 222 |
| 36 | The command <code>TikzEveryCell</code>                            | 225 |
| 37 | The command <code>\CreateBlocksInColumn</code>                    | 227 |
| 38 | The command <code>\ShowCellNames</code>                           | 227 |
| 39 | We process the options at package loading                         | 229 |
| 40 | About the package underscore                                      | 231 |
| 41 | Compatibility with <code>threeparttable</code>                    | 231 |
| 42 | Error messages of the package                                     | 231 |