

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

January 18, 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 7.5a of `nicematrix`, at the date of 2026/01/05.

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     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
35 }

```

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

```

36 \cs_new_protected:Npn \@@_error_or_warning:n
37 {
38   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
39     { \@@_warning:n }
40     { \@@_error:n }
41 }

```

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".

```

42 \bool_new:N \g_@@_messages_for_Overleaf_bool
43 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
44 {
45   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
46   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
47 }

48 \@@_msg_new:nn { mdwtab-loaded }
49 {
50   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
51   This~error~is~fatal.
52 }

53 \hook_gput_code:nnn { begindocument / end } { . }
54 { \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`).

```

55 \cs_new_protected:Npn \@@_collect_options:n #1
56 {
57   \peek_meaning:NTF [
58     { \@@_collect_options:nw { #1 } }
59     { #1 { } }
60   }

```

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

```

61 \NewDocumentCommand \@@_collect_options:nw { m r[] }
62 { \@@_collect_options:nn { #1 } { #2 } }
63
64 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
65 {
66   \peek_meaning:NTF [
67     { \@@_collect_options:nnw { #1 } { #2 } }
68     { #1 { #2 } }
69   }
70
71 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
72 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

## 3 Technical definitions

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

```

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

81 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
82 \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).

```

83 \tl_new:N \l_@@_argspec_tl

```

```

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

92 \hook_gput_code:nnn { begindocument } { . }
93 {
94   \IfPackageLoadedTF { tikz }
95   {

```

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.

```

96   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
97   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
98 }
99 {
100   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
101   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
102 }
103 }

```

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`).

```

104 \IfClassLoadedTF { revtex4-1 }
105 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
106 {
107   \IfClassLoadedTF { revtex4-2 }
108   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
109   {

```

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

```

110   \cs_if_exist:NT \rvtx@ifformat@geq
111   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
113 }
114 }

```

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.

```

115 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
116 {
117   \iow_now:Nn \@mainaux
118   {
119     \ExplSyntaxOn
120     \cs_if_free:NT \pgfsyspdfmark
121     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
122     \ExplSyntaxOff
123   }
124   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
125 }

```

We define a command `\iddots` similar to `\ddots` (`\ddots`) but with dots going forward (`\iddots`). 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.

```

126 \ProvideDocumentCommand \iddots { }
127 {
128   \mathinner
129   {
130     \mkern 1 mu
131     \box_move_up:nn { 1 pt } { \hbox { . } }
132     \mkern 2 mu
133     \box_move_up:nn { 4 pt } { \hbox { . } }
134     \mkern 2 mu
135     \box_move_up:nn { 7 pt }
136     { \vbox:n { \kern 7 pt \hbox { . } } }
137     \mkern 1 mu
138   }
139 }

```

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.

```

140 \hook_gput_code:nnn { begindocument } { . }
141 {
142   \IfPackageLoadedT { booktabs }
143   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
144 }
145 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
146 {
147   \let @@_old_pgfutil@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`).

```

148   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
149   {

```

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

```

150     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
151     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
152   }
153 }

```

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`.

```

154 \hook_gput_code:nnn { begindocument } { . }
155 {
156   \cs_set_protected:Npe @@_everycr:
157   {
158     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
159     { \noalign { \@@_in_everycr: } }
160   }
161   \IfPackageLoadedTF { colortbl }
162   {
163     \cs_set_eq:NN @@_old_cellcolor: \cellcolor
164     \cs_set_eq:NN @@_old_rowcolor: \rowcolor
165     \cs_new_protected:Npn @@_revert_colortbl:
166     {
167       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
168       {
169         \cs_set_eq:NN \cellcolor @@_old_cellcolor:
170         \cs_set_eq:NN \rowcolor @@_old_rowcolor:

```

```

171     }
172 }

```

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).

```

173 \cs_new_protected:Npn \@@_replace_columncolor:
174 {
175     \tl_replace_all:Nnn \g_@@_array_preamble_tl
176     { \columncolor }
177     { \@@_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.

```

178     }
179 }
180 {
181     \cs_new_protected:Npn \@@_revert_colortbl: { }
182     \cs_new_protected:Npn \@@_replace_columncolor:
183     { \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.

```

184 \def \CT@arc@ { }
185 \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
186 \def \CT@arc@ #1 #2
187 {
188     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
189     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
190 }

```

Idem for `\CT@drs@`.

```

191 \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
192 \def \CT@drs@ #1 #2
193 {
194     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
195     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
196 }
197 \def \hline
198 {
199     \noalign { \ifnum 0 = ` } \fi
200     \cs_set_eq:NN \hskip \vskip
201     \cs_set_eq:NN \vrule \hrule
202     \cs_set_eq:NN \@width \@height
203     { \CT@arc@ \vline }
204     \futurelet \reserved@a
205     \@xhline
206 }
207 }
208 }

```

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.

```

209 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
210 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
211 {
212     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
213     \int_compare:nNnT { #1 } > { \c_one_int }
214     { \multispan { \int_eval:n { #1 - 1 } } & }
215     \multispan { \int_eval:n { #2 - #1 + 1 } }
216     {
217         \CT@arc@
218         \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>

```
219     \skip_horizontal:N \c_zero_dim
220 }
```

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`.

```
221     \everycr { }
222     \cr
223     \noalign { \skip_vertical:n { - \arrayrulewidth } }
224 }
```

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.

```
225 \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`.

```
226 { \@@_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*.

```
227 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
228 \cs_generate_variant:Nn \@@_cline_i:nn { e }
229 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
230 {
231     \tl_if_empty:nTF { #3 }
232     { \@@_cline_iii:w #1|#2-#2 \q_stop }
233     { \@@_cline_ii:w #1|#2-#3 \q_stop }
234 }
235 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
236 { \@@_cline_iii:w #1|#2-#3 \q_stop }
237 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
238 {
```

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).

```
239     \int_compare:nNnT { #1 } < { #2 }
240     { \multispan { \int_eval:n { #2 - #1 } } & }
241     \multispan { \int_eval:n { #3 - #2 + 1 } }
242     {
243         \CT@arc@
244         \leaders \hrule \@height \arrayrulewidth \hfill
245         \skip_horizontal:N \c_zero_dim
246     }
```

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

```
247     \peek_meaning_remove_ignore_spaces:NNTF \cline
248     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
249     { \everycr { } \cr }
250 }
```

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

```
251 \cs_set:Nn \@@_math_toggle: { $ } % $
```

---

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

```

252 \cs_new_protected:Npn \@@_set_CTarc:n #1
253 {
254   \tl_if_blank:nF { #1 }
255   {
256     \tl_if_head_eq_meaning:nNTF { #1 } [
257       { \def \CT@arc@ { \color #1 } }
258       { \def \CT@arc@ { \color { #1 } } }
259     ]
260   }
261 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

262 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
263 {
264   \tl_if_head_eq_meaning:nNTF { #1 } [
265     { \def \CT@drsc@ { \color #1 } }
266     { \def \CT@drsc@ { \color { #1 } } }
267   ]

```

The following command must *not* be protected since it will be used to write instructions in the `\g_@@_pre_code_before_tl`.

```

268 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
269 {
270   \tl_if_head_eq_meaning:nNTF { #2 } [
271     { #1 #2 }
272     { #1 { #2 } }
273   ]
274 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

275 \cs_new_protected:Npn \@@_color:n #1
276 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
277 \cs_generate_variant:Nn \@@_color:n { o }

```

```

278 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
279 {
280   \tl_set_rescan:Nno
281     #1
282     {
283       \char_set_catcode_other:N >
284       \char_set_catcode_other:N <
285     }
286   #1
287 }

```

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We create several more in the same spirit.

```

288 \dim_new:N \l_@@_tmpc_dim
289 \dim_new:N \l_@@_tmpd_dim
290 \tl_new:N \l_@@_tmpc_tl
291 \tl_new:N \l_@@_tmpd_tl
292 \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.

```
293 \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).

```
294 \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.

```
295 \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.

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

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

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

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

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

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

```
300 \bool_new:N \g_@@_delims_bool
301 \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.

```
302 \bool_new:N \l_@@_preamble_bool
303 \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.

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

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

```
305 \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.

```
306 \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).

```
307 \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.).

```
308 \dim_new:N \l_@@_col_width_dim
309 \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.

```
310 \int_new:N \g_@@_row_total_int
311 \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).

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

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

```
313 \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.

```
314 \tl_new:N \l_@@_hpos_cell_tl
315 \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`.

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

Idem for the mono-row blocks.

```
317 \dim_new:N \g_@@_blocks_ht_dim
318 \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}`).

```
319 \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`.

```
320 \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.

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

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

```
322 \bool_new:N \l_@@_notes_detect_duplicates_bool
323 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```

324 \bool_new:N \l_@@_initial_open_bool
325 \bool_new:N \l_@@_final_open_bool
326 \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.

```

327 \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).

```

328 \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).

```

329 \tl_new:N \l_@@_rule_color_tl

```

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

```

330 \bool_new:N \g_@@_rotate_bool

```

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

```

331 \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.

```

332 \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`).

```

333 \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`.

```

334 \bool_new:N \g_@@_V_of_X_bool

```

```

335 \bool_new:N \g_@@_caption_finished_bool

```

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

```

336 \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 }`).

```

337 \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).

```

338 \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.

```

339 \seq_new:N \g_@@_size_seq

```

```

340 \tl_new:N \g_@@_left_delim_tl

```

```

341 \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}`).

```
342 \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`).

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

For `\multicolumn`.

```
344 \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`.

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

```
346 \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 `..`.

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

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

```
349 \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`).

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

```
351 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
352 {
```

```
353   \if_mode_math: \else:
```

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

```
355   \fi:
```

```
356 }
```

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.

```
357 \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”.

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

```
359 \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`).

```
360 \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*.

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

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

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

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)`.

```

364 \cs_new:Npn \@@_full_name_env:
365 {
366   \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
367   { command \space \c_backslash_str \g_@@_name_env_str }
368   { environment \space \{ \g_@@_name_env_str \} }
369 }

```

```

370 \tl_new:N \g_@@_cell_after_hook_tl % 2025/03/22

```

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

```

371 \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.

```

372 \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.

```

373 \tl_new:N \g_@@_pre_code_before_tl
374 \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.

```

375 \tl_new:N \g_@@_pre_code_after_tl
376 \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.

```

377 \bool_new:N \l_@@_in_code_after_bool

```

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

```

378 \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.

```

379 \int_new:N \l_@@_old_iRow_int
380 \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).

```

381 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

382 \tl_new:N \l_@@_rules_color_tl

```

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

```
383 \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.

```
384 \bool_new:N \l_@@_X_columns_aux_bool
385 \dim_new:N \l_@@_X_columns_dim
```

```
386 \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.

```
387 \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).

```
388 \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).

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

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

```
391 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
403 \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”.

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

```
405 \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.

```
406 \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!).

```
407 \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.

```
408 \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}`.

```
409 \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.

```
410 \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`).

```
411 \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`).

```
412 \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.

```
413 \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).

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

```
415 \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.

```
416 \int_new:N \g_@@_ddots_int
417 \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.

```
418 \dim_new:N \g_@@_delta_x_one_dim
419 \dim_new:N \g_@@_delta_y_one_dim
420 \dim_new:N \g_@@_delta_x_two_dim
421 \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).

```
422 \int_new:N \l_@@_row_min_int
423 \int_new:N \l_@@_row_max_int
424 \int_new:N \l_@@_col_min_int
425 \int_new:N \l_@@_col_max_int

426 \int_new:N \l_@@_initial_i_int
427 \int_new:N \l_@@_initial_j_int
428 \int_new:N \l_@@_final_i_int
429 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
430 \int_new:N \l_@@_start_int
431 \int_set_eq:NN \l_@@_start_int \c_one_int
432 \int_new:N \l_@@_end_int
433 \int_new:N \l_@@_local_start_int
434 \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.

```
435 \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).

```
436 \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`.

```
437 \tl_new:N \l_@@_fill_tl
438 \tl_new:N \l_@@_opacity_tl
439 \tl_new:N \l_@@_draw_tl
440 \seq_new:N \l_@@_tikz_seq
441 \clist_new:N \l_@@_borders_clist
442 \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.

```
443 \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`.

```
444 \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.

```
445 \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.

```
446 \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`).

```
447 \str_new:N \l_@@_hpos_block_str
448 \str_set:Nn \l_@@_hpos_block_str { c }
449 \bool_new:N \l_@@_hpos_of_block_cap_bool
450 \bool_new:N \l_@@_p_block_bool
```

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

```
451 \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).

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

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

```
453 \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).

```
454 \bool_new:N \l_@@_vlines_block_bool
455 \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.

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

457 \dim_new:N \l_@@_submatrix_extra_height_dim
458 \dim_new:N \l_@@_submatrix_left_xshift_dim
459 \dim_new:N \l_@@_submatrix_right_xshift_dim
460 \clist_new:N \l_@@_hlines_clist
461 \clist_new:N \l_@@_vlines_clist
462 \clist_new:N \l_@@_submatrix_hlines_clist
463 \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}`).

```
464 \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.

```
465 \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`).

```
466 \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.

```
467 \int_new:N \l_@@_first_row_int
468 \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.

```
469 \int_new:N \l_@@_first_col_int
470 \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).

```
471 \int_new:N \l_@@_last_row_int
472 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
474 \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  $0$ .

```
475 \int_new:N \l_@@_last_col_int
476 \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.

```
477 \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`).

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

## Some utilities

```
479 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
480 {
```

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

```
481 \def \l_tmpa_tl { #1 }
482 \def \l_tmpb_tl { #2 }
483 }
```

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`.

```
484 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
485 {
486   \clist_if_in:NnF #1 { all }
487   {
488     \clist_clear:N \l_tmpa_clist
489     \clist_map_inline:Nn #1
490     {
491       \tl_if_head_eq_meaning:nNTF { ##1 } -
492       {
```

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.

```
493       \int_if_zero:nF { #2 }
494       {
495         \clist_put_right:Ne \l_tmpa_clist
496         { \int_eval:n { #2 + (##1) + 1 } }
497       }
498     }
499   }
```

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

```

500         \tl_if_in:nnTF { ##1 } { - }
501         { \@@_cut_on_hyphen:w ##1 \q_stop }
502         {

```

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

```

503         \def \l_tmpa_tl { ##1 }
504         \def \l_tmpb_tl { ##1 }
505     }
506     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
507     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
508 }
509 }
510 \tl_set_eq:NN #1 \l_tmpa_clist
511 }
512 }

```

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.

```

513 \hook_gput_code:nnn { begindocument } { . }
514 {
515     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
516     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
517 }

```

## 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.

```
518 \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`.

```
519 \int_new:N \g_@@_tabularnote_int
520 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

521 \seq_new:N \g_@@_notes_seq
522 \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.

```
523 \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.

```
524 \seq_new:N \l_@@_notes_labels_seq
525 \newcounter { nicematrix_draft }
526 \cs_new_protected:Npn \@@_notes_format:n #1
527 {
528   \setcounter { nicematrix_draft } { #1 }
529   \@@_notes_style:n { nicematrix_draft }
530 }
```

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

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

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

```
532 \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`.

```
533 \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`.

```
534 \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`).

```
535 \hook_gput_code:nnn { begindocument } { . }
536 {
537   \IfPackageLoadedTF { enumitem }
538   {
```

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.

```

539     \newlist { tabularnotes } { enumerate } { 1 }
540     \setlist [ tabularnotes ]
541     {
542         topsep = \c_zero_dim ,
543         noitemsep ,
544         leftmargin = * ,
545         align = left ,
546         labelsep = \c_zero_dim ,
547         label =
548             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
549     }
550     \newlist { tabularnotes* } { enumerate* } { 1 }
551     \setlist [ tabularnotes* ]
552     {
553         afterlabel = \nobreak ,
554         itemjoin = \quad ,
555         label =
556             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
557     }

```

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`.

```

558     \NewDocumentCommand \tabularnote { o m }
559     {
560         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
561         {
562             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
563             { \@@_error:n { tabularnote~forbidden } }
564             {
565                 \bool_if:NTF \l_@@_in_caption_bool
566                 \@@_tabularnote_caption:nn
567                 \@@_tabularnote:nn
568                 { #1 } { #2 }
569             }
570         }
571     }
572 }
573 {
574     \NewDocumentCommand \tabularnote { o m }
575     { \@@_err_enumitem_not_loaded: }
576 }
577 }

578 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
579 {
580     \@@_error_or_warning:n { enumitem~not~loaded }
581     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
582 }

583 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
584 { \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`.

```

585 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
586 {

```

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).

```

587     \int_zero:N \l_tmpa_int
588     \bool_if:NT \l_@@_notes_detect_duplicates_bool
589     {

```

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`.

```

590     \int_zero:N \l_tmpb_int
591     \seq_map_indexed_inline:Nn \g_@@_notes_seq
592     {
593         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
594         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 } }
595         {
596             \tl_if_novalue:nTF { #1 }
597             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
598             { \int_set:Nn \l_tmpa_int { ##1 } }
599             \seq_map_break:
600         }
601     }
602     \int_if_zero:nF { \l_tmpa_int }
603     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
604 }
605 \int_if_zero:nT { \l_tmpa_int }
606 {
607     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
608     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
609 }
610 \seq_put_right:Ne \l_@@_notes_labels_seq
611 {
612     \tl_if_novalue:nTF { #1 }
613     {
614         \@@_notes_format:n
615         {
616             \int_eval:n
617             {
618                 \int_if_zero:nTF { \l_tmpa_int }
619                 { \c@tabularnote }
620                 { \l_tmpa_int }
621             }
622         }
623     }
624     { #1 }
625 }
626 \peek_meaning:NF \tabularnote
627 {

```

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`.

```

628     \hbox_set:Nn \l_tmpa_box
629     {

```

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

```

630         \@@_notes_label_in_tabular:n
631         { \seq_use:Nn \l_@@_notes_labels_seq { , } }
632     }

```

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

```

633     \int_gdecr:N \c@tabularnote
634     \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.

```

635     \int_gincr:N \g_@@_tabularnote_int
636     \refstepcounter { tabularnote }
637     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
638     { \int_gincr:N \c@tabularnote }
639     \seq_clear:N \l_@@_notes_labels_seq
640     \bool_lazy_or:nnTF
641     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
642     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
643     {
644         \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).

```

645         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
646     }
647     { \box_use:N \l_tmpa_box }
648 }
649 }

```

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`.

```

650 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
651 {
652     \bool_if:NTF \g_@@_caption_finished_bool
653     {
654         \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
655         { \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`!

```

656     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
657     { \@@_error:n { Identical-notes-in-caption } }
658 }
659 {

```

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

```

660     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
661     {

```

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.

```

662         \bool_gset_true:N \g_@@_caption_finished_bool
663         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
664         \int_gzero:N \c@tabularnote
665     }

```



```

666         { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
667     }

```

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!

```

668     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
669     \seq_put_right:Ne \l_@@_notes_labels_seq
670     {
671         \tl_if_novalue:nTF { #1 }
672         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
673         { #1 }
674     }
675     \peek_meaning:NF \tabularnote
676     {
677         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
678         \seq_clear:N \l_@@_notes_labels_seq
679     }
680 }

```

```

681 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
682 { \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.

```

683 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
684 {
685     \begin { pgfscope }
686     \pgfset
687     {
688         inner~sep = \c_zero_dim ,
689         minimum~size = \c_zero_dim
690     }
691     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
692     \pgfnode
693     { rectangle }
694     { center }
695     {
696         \vbox_to_ht:nn
697         { \dim_abs:n { #5 - #3 } }
698         {
699             \vfill
700             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { } { }
701         }
702     }
703     { #1 }
704     { }
705     \end { pgfscope }
706 }

```

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.

```

707 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
708 {
709     \begin { pgfscope }
710     \pgfset
711     {
712         inner~sep = \c_zero_dim ,

```

```

713     minimum-size = \c_zero_dim
714   }
715   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
716   \pgfpointdiff { #3 } { #2 }
717   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
718   \pgfnode
719     { rectangle }
720     { center }
721     {
722       \vbox_to_ht:nn
723         { \dim_abs:n \l_tmpb_dim }
724         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
725     }
726     { #1 }
727     { }
728   \end { pgfscope }
729 }

```

## 7 The options

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

```

730 \tl_new:N \l_@@_caption_tl
731 \tl_new:N \l_@@_short_caption_tl
732 \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).

```

733 \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`.

```

734 \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`).

```

735 \dim_new:N \l_@@_cell_space_top_limit_dim
736 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

737 \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.

```

738 \dim_new:N \l_@@_xdots_inter_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \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).

```

741 \dim_new:N \l_@@_xdots_shorten_start_dim
742 \dim_new:N \l_@@_xdots_shorten_end_dim
743 \hook_gput_code:nnn { begindocument } { . }
744 {
745   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
746   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
747 }
```

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.

```

748 \dim_new:N \l_@@_xdots_radius_dim
749 \hook_gput_code:nnn { begindocument } { . }
750 { \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.

```

751 \tl_new:N \l_@@_xdots_line_style_tl
752 \tl_const:Nn \c_@@_standard_tl { standard }
753 \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`.

```

754 \bool_new:N \l_@@_light_syntax_bool
755 \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).

```

756 \tl_new:N \l_@@_baseline_tl
757 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```

758 \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`).

```

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

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

```

760 \bool_new:N \l_@@_parallelize_diags_bool
761 \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.

```

762 \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.).

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

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

```
764 \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).

```
765 \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!

```
766 \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.

```
767 \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”.

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

```
769 \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).

```
770 \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`).

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

```
772 \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`.

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

```
774 \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.

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

```
776 \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 “:”.

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

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

```
778 \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.

```

779 \bool_new:N \l_@@_delimiters_max_width_bool

780 \keys_define:nn { nicematrix / xdots }
781 {
782   nullify .bool_set:N = \l_@@_nullify_dots_bool ,
783   nullify .default:n = true ,
784   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
785   brace-shift .value_required:n = true ,
786   brace-shift+ .code:n =
787     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
788   brace-shift+ .value_required:n = true ,
789   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
790   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
791   shorten-start .code:n =
792     \hook_gput_code:nnn { begindocument } { . }
793     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
794   shorten-start .value_required:n = true ,
795   shorten-start+ .code:n =
796     \hook_gput_code:nnn { begindocument } { . }
797     { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
798   shorten-start~+ .meta:n = { shorten-start+ = #1 } ,
799   shorten-start+ .value_required:n = true ,
800   shorten-end .code:n =
801     \hook_gput_code:nnn { begindocument } { . }
802     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
803   shorten-end .value_required:n = true ,
804   shorten-end+ .code:n =
805     \hook_gput_code:nnn { begindocument } { . }
806     { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
807   shorten-end+ .value_required:n = true ,
808   shorten-end~+ .meta:n = { shorten-end+ = #1 } ,
809   shorten .code:n =
810     \hook_gput_code:nnn { begindocument } { . }
811     {
812       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
813       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
814     } ,
815   shorten .value_required:n = true ,
816   shorten+ .code:n =
817     \hook_gput_code:nnn { begindocument } { . }
818     {
819       \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
820       \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
821     } ,
822   shorten~+ .meta:n = { shorten+ = #1 } ,
823   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
824   horizontal-labels .default:n = true ,
825   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
826   horizontal-label .default:n = true ,
827   line-style .code:n =
828     {
829       \bool_lazy_or:nnTF
830         { \cs_if_exist_p:N \tikzpicture }
831         { \str_if_eq_p:nn { #1 } { standard } }
832         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }

```

```

833     { \@@_error:n { bad-option-for-line-style } }
834   } ,
835   line-style .value_required:n = true ,
836   color .tl_set:N = \l_@@_xdots_color_tl ,
837   color .value_required:n = true ,
838   radius .code:n =
839     \hook_gput_code:nnn { begindocument } { . }
840     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
841   radius .value_required:n = true ,
842   inter .code:n =
843     \hook_gput_code:nnn { begindocument } { . }
844     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
845   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  $\wedge$ ,  $\_$  and  $\cdot$ . 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 `\wedge{...}`.

```

846   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
847   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
848   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).

```

849   draw-first .code:n = \prg_do_nothing: ,
850   unknown .code:n =
851     \@@_unknown_key:nn { nicematrix / xdots } { Unknown-key-for-xdots }
852 }

```

```

853 \keys_define:nn { nicematrix / rules }
854 {
855   color .tl_set:N = \l_@@_rules_color_tl ,
856   color .value_required:n = true ,
857   width .dim_set:N = \arrayrulewidth ,
858   width .value_required:n = true ,
859   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
860 }

```

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

```

861 \keys_define:nn { nicematrix / Global }
862 {
863   caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
864   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
865   color-inside .code:n = \@@_fatal:n { key-color-inside } ,
866   colortbl-like .code:n = \@@_fatal:n { key-color-inside } ,
867   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
868   ampersand-in-blocks .default:n = true ,
869   &-in-blocks .meta:n = ampersand-in-blocks ,
870   no-cell-nodes .code:n =
871     \bool_set_true:N \l_@@_no_cell_nodes_bool
872     \cs_set_protected:Npn \@@_node_cell:
873       { \set@color \box_use_drop:N \l_@@_cell_box } ,
874   no-cell-nodes .value_forbidden:n = true ,
875   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
876   rounded-corners .default:n = 4 pt ,
877   custom-line .code:n = \@@_custom_line:n { #1 } ,
878   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
879   rules .value_required:n = true ,
880   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
881   standard-cline .default:n = true ,
882   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,

```

```

883 cell-space-top-limit .value_required:n = true ,
884 cell-space-top-limit+ .code:n =
885   \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
886 cell-space-top-limit+ .value_required:n = true ,
887 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
888 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
889 cell-space-bottom-limit .value_required:n = true ,
890 cell-space-bottom-limit+ .code:n =
891   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
892 cell-space-bottom-limit+ .value_required:n = true ,
893 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
894 cell-space-limits .meta:n =
895   {
896     cell-space-top-limit = #1 ,
897     cell-space-bottom-limit = #1 ,
898   } ,
899 cell-space-limits .value_required:n = true ,
900 cell-space-limits+ .meta:n =
901   {
902     cell-space-top-limit += #1 ,
903     cell-space-bottom-limit += #1 ,
904   } ,
905 cell-space-limits+ .value_required:n = true ,
906 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
907 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
908 light-syntax .code:n =
909   \bool_set_true:N \l_@@_light_syntax_bool
910   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
911 light-syntax .value_forbidden:n = true ,
912 light-syntax-expanded .code:n =
913   \bool_set_true:N \l_@@_light_syntax_bool
914   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
915 light-syntax-expanded .value_forbidden:n = true ,
916 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
917 end-of-row .value_required:n = true ,
918
919 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
920 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
921 last-row .int_set:N = \l_@@_last_row_int ,
922 last-row .default:n = -1 ,
923
924 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
925 code-for-first-col .value_required:n = true ,
926 code-for-first-col+ .code:n =
927   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
928 code-for-first-col+ .value_required:n = true ,
929 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
930
931 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
932 code-for-last-col .value_required:n = true ,
933 code-for-last-col+ .code:n =
934   { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
935 code-for-last-col+ .value_required:n = true ,
936 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
937
938 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
939 code-for-first-row .value_required:n = true ,
940 code-for-first-row+ .code:n =
941   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
942 code-for-first-row+ .value_required:n = true ,
943 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
944
945 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,

```

```

946 code-for-last-row .value_required:n = true ,
947 code-for-last-row+ .code:n =
948   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
949 code-for-last-row+ .value_required:n = true ,
950 code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
951
952 hlines .clist_set:N = \l_@@_hlines_clist ,
953 vlines .clist_set:N = \l_@@_vlines_clist ,
954 hlines .default:n = all ,
955 vlines .default:n = all ,
956 vlines-in-sub-matrix .code:n =
957   {
958     \tl_if_single_token:nTF { #1 }
959     {
960       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
961       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

962     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
963   }
964   { \@@_error:n { One-letter-allowed } }
965 } ,
966 vlines-in-sub-matrix .value_required:n = true ,
967 hvlines .code:n =
968   {
969     \bool_set_true:N \l_@@_hvlines_bool
970     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
971     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
972   } ,
973 hvlines .value_forbidden:n = true ,
974 hvlines-except-borders .code:n =
975   {
976     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
977     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
978     \bool_set_true:N \l_@@_hvlines_bool
979     \bool_set_true:N \l_@@_except_borders_bool
980   } ,
981 hvlines-except-borders .value_forbidden:n = true ,
982 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.

```

983 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
984 renew-dots .value_forbidden:n = true ,
985 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
986 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
987 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
988 create-extra-nodes .meta:n =
989   { create-medium-nodes , create-large-nodes } ,
990 left-margin .dim_set:N = \l_@@_left_margin_dim ,
991 left-margin .default:n = \arraycolsep ,
992 right-margin .dim_set:N = \l_@@_right_margin_dim ,
993 right-margin .default:n = \arraycolsep ,
994 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
995 margin .default:n = \arraycolsep ,
996 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
997 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
998 extra-margin .meta:n =
999   { extra-left-margin = #1 , extra-right-margin = #1 } ,
1000 extra-margin .value_required:n = true ,
1001 respect-arraystretch .code:n =
1002   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1003 respect-arraystretch .value_forbidden:n = true ,

```



```

1004   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1005   pgf-node-code .value_required:n = true
1006 }

```

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

```

1007 \keys_define:nn { nicematrix / environments }
1008 {
1009   corners .clist_set:N = \l_@@_corners_clist ,
1010   corners .default:n = { NW , SW , NE , SE } ,
1011   code-before .code:n =
1012   {
1013     \tl_if_empty:nF { #1 }
1014     {
1015       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1016       \bool_set_true:N \l_@@_code_before_bool
1017     }
1018   } ,
1019   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}`.

```

1020   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1021   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1022   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
1023   baseline .tl_set:N = \l_@@_baseline_tl ,
1024   baseline .value_required:n = true ,
1025   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)`.

```

1026   \str_if_eq:eeTF { #1 } { auto }
1027   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1028   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1029   columns-width .value_required:n = true ,
1030   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.

```

1031   \legacy_if:nF { measuring@ }
1032   {
1033     \str_set:Nn \l_@@_name_str { #1 }
1034     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1035     { \@@_err_duplicate_names:n { #1 } }
1036     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1037   } ,
1038   name .value_required:n = true ,
1039   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1040   code-after .value_required:n = true ,
1041 }

1042 \cs_set:Npn \@@_err_duplicate_names:n #1
1043 { \@@_error:nn { Duplicate-name } { #1 } }

1044 \keys_define:nn { nicematrix / notes }
1045 {
1046   para .bool_set:N = \l_@@_notes_para_bool ,
1047   para .default:n = true ,
1048   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1049   code-before .value_required:n = true ,
1050   code-before+ .code:n =
1051   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1052   code-before+ .value_required:n = true ,
1053   code-before~+ .code:n =

```

```

1054     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1055     code-before~+ .value_required:n = true ,
1056     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1057     code-after .value_required:n = true ,
1058     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1059     bottomrule .default:n = true ,
1060     style .cs_set:Np = \@@_notes_style:n #1 ,
1061     style .value_required:n = true ,
1062     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1063     label-in-tabular .value_required:n = true ,
1064     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1065     label-in-list .value_required:n = true ,
1066     enumitem-keys .code:n =
1067     {
1068         \hook_gput_code:nnn { begindocument } { . }
1069         {
1070             \IfPackageLoadedT { enumitem }
1071             { \setlist* [ tabularnotes ] { #1 } }
1072         }
1073     } ,
1074     enumitem-keys .value_required:n = true ,
1075     enumitem-keys-para .code:n =
1076     {
1077         \hook_gput_code:nnn { begindocument } { . }
1078         {
1079             \IfPackageLoadedT { enumitem }
1080             { \setlist* [ tabularnotes* ] { #1 } }
1081         }
1082     } ,
1083     enumitem-keys-para .value_required:n = true ,
1084     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1085     detect-duplicates .default:n = true ,
1086     unknown .code:n =
1087     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1088 }

1089 \keys_define:nn { nicematrix / delimiters }
1090 {
1091     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1092     max-width .default:n = true ,
1093     color .tl_set:N = \l_@@_delimiters_color_tl ,
1094     color .value_required:n = true ,
1095 }

```

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

```

1096 \keys_define:nn { nicematrix }
1097 {
1098     NiceMatrixOptions .inherit:n =
1099     { nicematrix / Global } ,
1100     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1101     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1102     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1103     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1104     SubMatrix / rules .inherit:n = nicematrix / rules ,
1105     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1106     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1107     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1108     NiceMatrix .inherit:n =
1109     {
1110         nicematrix / Global ,
1111         nicematrix / environments ,
1112     } ,

```

```

1113   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1114   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1115   NiceTabular .inherit:n =
1116   {
1117       nicematrix / Global ,
1118       nicematrix / environments
1119   } ,
1120   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1121   NiceTabular / rules .inherit:n = nicematrix / rules ,
1122   NiceTabular / notes .inherit:n = nicematrix / notes ,
1123   NiceArray .inherit:n =
1124   {
1125       nicematrix / Global ,
1126       nicematrix / environments ,
1127   } ,
1128   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1129   NiceArray / rules .inherit:n = nicematrix / rules ,
1130   pNiceArray .inherit:n =
1131   {
1132       nicematrix / Global ,
1133       nicematrix / environments ,
1134   } ,
1135   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1136   pNiceArray / rules .inherit:n = nicematrix / rules ,
1137 }

```

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

```

1138 \keys_define:nn { nicematrix / NiceMatrixOptions }
1139 {
1140     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1141     delimiters / color .value_required:n = true ,
1142     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1143     delimiters / max-width .default:n = true ,
1144     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1145     delimiters .value_required:n = true ,
1146     width .dim_set:N = \l_@@_width_dim ,
1147     width .value_required:n = true ,
1148     last-col .code:n =
1149     \tl_if_empty:nF { #1 }
1150     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1151     \int_zero:N \l_@@_last_col_int ,
1152     small .bool_set:N = \l_@@_small_bool ,
1153     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.

```

1154     renew-matrix .code:n = \@@_renew_matrix: ,
1155     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}`.

```

1156     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.

```

1157     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)`.

```

1158     \str_if_eq:eeTF { #1 } { auto }
1159     { \@@_error:n { Option~auto~for~columns-width } }
1160     { \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.

```

1161   allow-duplicate-names .code:n =
1162     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1163   allow-duplicate-names .value_forbidden:n = true ,
1164   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1165   notes .value_required:n = true ,
1166   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1167   sub-matrix .value_required:n = true ,
1168   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1169   matrix / columns-type .value_required:n = true ,
1170   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1171   caption-above .default:n = true ,
1172   unknown .code:n =
1173     \@@_unknown_key:nn
1174       { nicematrix / Global , nicematrix / NiceMatrixOptions }
1175       { Unknown-key-for-NiceMatrixOptions }
1176 }

```

`\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.

```

1177 \NewDocumentCommand \NiceMatrixOptions { m }
1178 { \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.

```

1179 \keys_define:nn { nicematrix / NiceMatrix }
1180 {
1181   last-col .code:n = \tl_if_empty:nTF { #1 }
1182     {
1183       \bool_set_true:N \l_@@_last_col_without_value_bool
1184       \int_set:Nn \l_@@_last_col_int { -1 }
1185     }
1186     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1187   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1188   columns-type .value_required:n = true ,
1189   l .meta:n = { columns-type = l } ,
1190   r .meta:n = { columns-type = r } ,
1191   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1192   delimiters / color .value_required:n = true ,
1193   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1194   delimiters / max-width .default:n = true ,
1195   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1196   delimiters .value_required:n = true ,
1197   small .bool_set:N = \l_@@_small_bool ,
1198   small .value_forbidden:n = true ,
1199   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1200 }

```

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

```

1201 \keys_define:nn { nicematrix / NiceArray }
1202 {

```

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.

```

1203   small .bool_set:N = \l_@@_small_bool ,
1204   small .value_forbidden:n = true ,
1205   last-col .code:n = \tl_if_empty:nF { #1 }
1206     { \@@_error:n { last-col-non-empty-for-NiceArray } }

```

```

1207         \int_zero:N \l_@@_last_col_int ,
1208     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1209     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1210     unknown .code:n =
1211         \@@_unknown_key:nn
1212         { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
1213         { Unknown-key~for~NiceArray }
1214 }
1215 \keys_define:nn { nicematrix / pNiceArray }
1216 {
1217     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1218     last-col .code:n = \tl_if_empty:nF { #1 }
1219         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1220         \int_zero:N \l_@@_last_col_int ,
1221     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1222     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1223     delimiters / color .value_required:n = true ,
1224     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1225     delimiters / max-width .default:n = true ,
1226     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1227     delimiters .value_required:n = true ,
1228     small .bool_set:N = \l_@@_small_bool ,
1229     small .value_forbidden:n = true ,
1230     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1231     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1232     unknown .code:n =
1233         \@@_unknown_key:nn
1234         { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1235         { Unknown-key~for~NiceMatrix }
1236 }

```

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

```

1237 \keys_define:nn { nicematrix / NiceTabular }
1238 {

```

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.

```

1239     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1240         \bool_set_true:N \l_@@_width_used_bool ,
1241     width .value_required:n = true ,
1242     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1243     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1244     tabularnote .value_required:n = true ,
1245     caption .tl_set:N = \l_@@_caption_tl ,
1246     caption .value_required:n = true ,
1247     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1248     short-caption .value_required:n = true ,
1249     label .tl_set:N = \l_@@_label_tl ,
1250     label .value_required:n = true ,
1251     last-col .code:n = \tl_if_empty:nF { #1 }
1252         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1253         \int_zero:N \l_@@_last_col_int ,
1254     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1255     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1256     unknown .code:n =
1257         \@@_unknown_key:nn
1258         { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1259         { Unknown-key~for~NiceTabular }
1260 }

```

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
```

```
1261 \keys_define:nn { nicematrix / CodeAfter }
1262 {
1263   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1264   delimiters / color .value_required:n = true ,
1265   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1266   rules .value_required:n = true ,
1267   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1268   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1269   sub-matrix .value_required:n = true ,
1270   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1271 }
```

## 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}`).

```
1272 \cs_new_protected:Npn \@@_cell_begin:
1273 {
```

`\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.

```
1274   \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).

```
1275   \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.

```
1276   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1277   \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.

```
1278   \if_int_compare:w \c@jCol = \c_one_int
1279     \if_int_compare:w \l_@@_first_col_int = \c_one_int
1280       \@@_begin_of_row:
1281     \fi:
1282   \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:`.

```
1283   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1284 \@@_tuning_not_tabular_begin:
1285 \@@_tuning_first_row:
1286 \@@_tuning_last_row:
1287 \g_@@_row_style_tl
1288 }

```

The following command will be nullified unless there is a first row.  
Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

1289 \cs_new_protected:Npn \@@_tuning_first_row:
1290 {
1291   \if_int_compare:w \c@iRow = \c_zero_int
1292     \if_int_compare:w \c@jCol > \c_zero_int
1293       \l_@@_code_for_first_row_tl
1294       \xglobal \colorlet { nicematrix-first-row } { . }
1295   \fi:
1296   \fi:
1297 }

```

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:nNnT { \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.

```

1298 \cs_new_protected:Npn \@@_tuning_last_row:
1299 {
1300   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1301     \l_@@_code_for_last_row_tl
1302     \xglobal \colorlet { nicematrix-last-row } { . }
1303   \fi:
1304 }

```

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

```

1305 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1306 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1307 {
1308   \m@th
1309   $ \% $

```

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

```

1310 \@@_tuning_key_small:
1311 }
1312 \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.

```

1313 \cs_new_protected:Npn \@@_begin_of_row:
1314 {
1315   \int_gincr:N \c@iRow
1316   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1317   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1318   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1319   \pgfpicture
1320   \pgfrememberpicturepositiononpagetrue
1321   \pgfcoordinate
1322   { \@@_env: - row - \int_use:N \c@iRow - base }
1323   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1324   \str_if_empty:NF \l_@@_name_str
1325   {
1326     \pgfnodealias
1327     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1328     { \@@_env: - row - \int_use:N \c@iRow - base }
1329   }
1330   \endpgfpicture
1331 }

```

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:nNnT { \c@iRow } = { 1 }
    {
      \dim_compare:nNnT
      { \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.

```

1332 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1333 {
1334   \if_int_compare:w \c@iRow = \c_zero_int
1335     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1336       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1337     \fi:
1338     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1339       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1340     \fi:
1341   \else:
1342     \if_int_compare:w \c@iRow = \c_one_int

```



```

1343     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1344     \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1345     \fi:
1346     \fi:
1347     \fi:
1348   }

1349 \cs_new_protected:Npn \@@_rotate_cell_box:
1350 {
1351   \box_rotate:Nn \l_@@_cell_box { 90 }
1352   \bool_if:NTF \g_@@_rotate_c_bool
1353   {
1354     \hbox_set:Nn \l_@@_cell_box
1355     {
1356       \m@th
1357       $ % $
1358       \vcenter { \box_use:N \l_@@_cell_box }
1359       $ % $
1360     }
1361   }
1362   {
1363     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1364     {
1365       \vbox_set_top:Nn \l_@@_cell_box
1366       {
1367         \vbox_to_zero:n { }
1368         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1369         \box_use:N \l_@@_cell_box
1370       }
1371     }
1372   }
1373   \bool_gset_false:N \g_@@_rotate_bool
1374   \bool_gset_false:N \g_@@_rotate_c_bool
1375 }

```

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:nNnT { \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.

```

1376 \cs_set_protected:Npn \@@_adjust_size_box:
1377 {
1378   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim

```

```

1379     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1380     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1381     \fi:
1382     \global \g_@@_blocks_wd_dim = \c_zero_dim
1383   \fi:
1384   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1385     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1386     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1387     \fi
1388     \global \g_@@_blocks_dp_dim = \c_zero_dim
1389   \fi:
1390   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1391     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1392     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1393     \fi:
1394     \global \g_@@_blocks_ht_dim = \c_zero_dim
1395   \fi:
1396 }
1397 \cs_new_protected:Npn \@@_cell_end:
1398 {

```

The following command is nullified in the tabulars.

```

1399   \@@_tuning_not_tabular_end:
1400   \hbox_set_end:
1401   \@@_cell_end_i:
1402 }
1403 \cs_new_protected:Npn \@@_cell_end_i:
1404 {

```

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.

```

1405   \g_@@_cell_after_hook_tl
1406   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1407   \@@_adjust_size_box:
1408   \box_set_ht:Nn \l_@@_cell_box
1409   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1410   \box_set_dp:Nn \l_@@_cell_box
1411   { \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”).

```

1412   \@@_update_max_cell_width:

```

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

```

1413   \@@_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.

```

1414 \bool_if:NTF \g_@@_empty_cell_bool
1415 { \box_use_drop:N \l_@@_cell_box }
1416 {
1417   \bool_if:NTF \g_@@_not_empty_cell_bool
1418   { \@@_print_node_cell: }
1419   {
1420     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1421     { \@@_print_node_cell: }
1422     { \box_use_drop:N \l_@@_cell_box }
1423   }
1424 }
1425 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1426 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1427 \bool_gset_false:N \g_@@_empty_cell_bool
1428 \bool_gset_false:N \g_@@_not_empty_cell_bool
1429 }

```

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

```

1430 \cs_new_protected:Npn \@@_update_max_cell_width:
1431 {
1432   \dim_gset:Nn \g_@@_max_cell_width_dim
1433   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1434 }

```

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`).

```

1435 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1436 {
1437   \@@_math_toggle:
1438   \hbox_set_end:
1439   \bool_if:NF \g_@@_rotate_bool
1440   {
1441     \hbox_set:Nn \l_@@_cell_box
1442     {
1443       \makebox [ \l_@@_col_width_dim ] [ s ]
1444       { \hbox_unpack_drop:N \l_@@_cell_box }
1445     }
1446   }
1447   \@@_cell_end_i:
1448 }

```

```

1449 \pgfset
1450 {
1451   nicematrix / cell-node /.style =
1452   {
1453     inner~sep = \c_zero_dim ,
1454     minimum~width = \c_zero_dim
1455   }
1456 }

```

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.

```

1457 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1458 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1459 {
1460   \use:c
1461   {
1462     __siunitx_table_align_
1463     \bool_if:NTF \l__siunitx_table_text_bool
1464       { \l__siunitx_table_align_text_tl }
1465       { \l__siunitx_table_align_number_tl }
1466     :n
1467   }
1468   { #1 }
1469 }

```

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!

```

1470 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1471 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1472 {
1473   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1474   \hbox:n
1475   {
1476     \pgfsys@markposition
1477     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1478   }
1479   #1
1480   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1481   \hbox:n
1482   {
1483     \pgfsys@markposition
1484     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1485   }
1486 }

1487 \cs_new_protected:Npn \@@_print_node_cell:
1488 {
1489   \socket_use:nn { nicematrix / siunitx-wrap }
1490   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1491 }

```

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

```

1492 \cs_new_protected:Npn \@@_node_cell:
1493 {
1494   \pgfpicture
1495   \pgfsetbaseline \c_zero_dim
1496   \pgfrememberpicturepositiononpagetrue
1497   \pgfset { nicematrix / cell-node }
1498   \pgfnode
1499   { rectangle }
1500   { base }
1501   {

```

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).

```

1502   \sys_if_engine_xetex:T { \set@color }
1503   \box_use:N \l_@@_cell_box
1504 }
1505 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1506 { \l_@@_pgf_node_code_tl }

```

```

1507 \str_if_empty:NF \l_@@_name_str
1508 {
1509     \pgfnodealias
1510     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1511     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1512 }
1513 \endpgfpicture
1514 }

```

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}

```

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

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).

```

1515 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1516 {
1517     \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1518     { g_@@_ #2 _ lines _ tl }
1519     {
1520         \use:c { @@ _ draw _ #2 : nnn }
1521         { \int_use:N \c@iRow }
1522         { \int_use:N \c@jCol }
1523         { \exp_not:n { #3 } }
1524     }
1525 }

1526 \cs_new_protected:Npn \@@_array:n
1527 {
1528     \dim_set:Nn \col@sep
1529     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1530     \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1531     { \def \@halignto { } }
1532     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1533 \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. Remark 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)`.

```

1534 [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1535 }
1536 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\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`.

```
1537 \bool_if:NTF \c_@@_revtex_bool
1538 { \cs_set_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`.

```
1539 { \cs_set_eq:cN { \@@_old_ar@ialign: } \ar@ialign }
```

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

```
1540 \cs_new_protected:Npn \@@_create_row_node:
1541 {
1542   \int_compare:nNt { \c@iRow } > { \g_@@_last_row_node_int }
1543   {
1544     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1545     \@@_create_row_node_i:
1546   }
1547 }
1548 \cs_new_protected:Npn \@@_create_row_node_i:
1549 {
```

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

```
1550   \hbox
1551   {
1552     \bool_if:NT \l_@@_code_before_bool
1553     {
1554       \vtop
1555       {
1556         \skip_vertical:N 0.5\arrayrulewidth
1557         \pgfsys@markposition
1558         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1559         \skip_vertical:N -0.5\arrayrulewidth
1560       }
1561     }
1562     \pgfpicture
1563     \pgfrememberpicturepositiononpagetrue
1564     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1565     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1566     \str_if_empty:NF \l_@@_name_str
1567     {
1568       \pgfnodealias
1569       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1570       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1571     }
1572     \endpgfpicture
1573   }
1574 }
```

```
1575 \cs_new_protected:Npn \@@_in_everycr:
1576 {
1577   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1578   \tbl_update_cell_data_for_next_row:
1579   \int_gzero:N \c@jCol
1580   \bool_gset_false:N \g_@@_after_col_zero_bool
1581   \bool_if:NF \g_@@_row_of_col_done_bool
1582   {
1583     \@@_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).

```
1584   \clist_if_empty:NF \l_@@_hlines_clist
1585   {
```

```

1586     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1587     {
1588         \clist_if_in:NiT
1589         \l_@@_hlines_clist
1590         { \int_eval:n { \c@iRow + 1 } }
1591     }
1592     {

```

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.

```

1593         \int_compare:nNnT { \c@iRow } > { -1 }
1594         {
1595             \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1596             { \hrule height \arrayrulewidth width \c_zero_dim }
1597         }
1598     }
1599 }
1600 }
1601 }

```

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

```

1602 \cs_set_protected:Npn \@@_renew_dots:
1603 {
1604     \cs_set_eq:NN \ldots \@@_Ldots:
1605     \cs_set_eq:NN \cdots \@@_Cdots:
1606     \cs_set_eq:NN \vdots \@@_Vdots:
1607     \cs_set_eq:NN \ddots \@@_Ddots:
1608     \cs_set_eq:NN \iddots \@@_Iddots:
1609     \cs_set_eq:NN \dots \@@_Ldots:
1610     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1611 }

```

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>.

```

1612 \hook_gput_code:nnn { begindocument } { . }
1613 {
1614     \IfPackageLoadedTF { booktabs }
1615     {
1616         \cs_new_protected:Npn \@@_patch_booktabs:
1617         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1618     }
1619     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1620 }

```

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`.

```

1621 \cs_new_protected:Npn \@@_some_initialization:
1622 {

```

<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}`.

```

1623 \@@_everycr:
1624 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1625 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1626 \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1627 \dim_gzero:N \g_@@_dp_ante_last_row_dim
1628 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1629 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1630 }

```

`\@@_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`.

```

1631 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1632 {

```

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 mains blocks will be added in `\g_@@_pos_of_blocks_seq` during the construction of the array.

```

1633 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1634 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1635 \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.

```

1636 \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.

```

1637 \fp_gzero:N \g_@@_total_X_weight_fp
1638 \bool_gset_false:N \g_@@_V_of_X_bool

1639 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1640 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1641 \@@_patch_booktabs:
1642 \box_clear_new:N \l_@@_cell_box
1643 \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}`).

```

1644 \bool_if:NT \l_@@_small_bool
1645 {
1646     \def \arraystretch { 0.47 }
1647     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1648 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1649 }

```



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!

```

1650   \bool_if:NT \g_@@_create_cell_nodes_bool
1651   {
1652     \tl_put_right:Nn \@@_begin_of_row:
1653     {
1654       \pgfsys@markposition
1655       { \@@_env: - row - \int_use:N \c@iRow - base }
1656     }
1657     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1658   }

```

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`.

```

1659   \bool_if:NF \c_@@_revtex_bool
1660   {
1661     \def \ar@ialign
1662     {
1663       \tbl_init_cell_data_for_table:
1664       \@@_some_initialization:
1665       \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this 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`.

```

1666       \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1667       \halign
1668     }
1669   }

```

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).

```

1670   \bool_if:NT \c_@@_revtex_bool
1671   {
1672     \IfPackageLoadedT { colortbl }
1673     { \cs_set_protected:Npn \CT@setup { } }
1674   }

```

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).

```

1675   \cs_set_eq:NN \@@_old_ldots: \ldots
1676   \cs_set_eq:NN \@@_old_cdots: \cdots
1677   \cs_set_eq:NN \@@_old_vdots: \vdots
1678   \cs_set_eq:NN \@@_old_ddots: \ddots
1679   \cs_set_eq:NN \@@_old_iddots: \iddots
1680   \bool_if:NTF \l_@@_standard_cline_bool
1681   { \cs_set_eq:NN \cline \@@_standard_cline: }
1682   { \cs_set_eq:NN \cline \@@_cline: }
1683   \cs_set_eq:NN \Ldots \@@_Ldots:
1684   \cs_set_eq:NN \Cdots \@@_Cdots:
1685   \cs_set_eq:NN \Vdots \@@_Vdots:
1686   \cs_set_eq:NN \Ddots \@@_Ddots:
1687   \cs_set_eq:NN \Iddots \@@_Iddots:
1688   \cs_set_eq:NN \Hline \@@_Hline:
1689   \cs_set_eq:NN \Hspace \@@_Hspace:
1690   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:

```

```

1691 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1692 \cs_set_eq:NN \Block \@@_Block:
1693 \cs_set_eq:NN \rotate \@@_rotate:
1694 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1695 \cs_set_eq:NN \dotfill \@@_dotfill:
1696 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1697 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1698 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1699 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1700 \cs_set_eq:NN \TopRule \@@_TopRule
1701 \cs_set_eq:NN \MidRule \@@_MidRule
1702 \cs_set_eq:NN \BottomRule \@@_BottomRule
1703 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1704 \cs_set_eq:NN \Hbrace \@@_Hbrace
1705 \cs_set_eq:NN \Vbrace \@@_Vbrace
1706 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1707 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1708 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1709 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1710 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1711 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1712 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1713 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1714 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1715 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1716 \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:`.

```

1717 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1718 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1719 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1720 \@@_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!).

```

1721 \tl_if_exist:NT \l_@@_note_in_caption_tl
1722 {
1723   \tl_if_empty:NF \l_@@_note_in_caption_tl
1724   {
1725     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1726     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1727   }
1728 }

```

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).

```

1729 \seq_gclear:N \g_@@_multicolumn_cells_seq
1730 \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).

```

1731 \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`).

```

1732 \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.

```

1733 \int_gzero:N \g_@@_col_total_int
1734 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1735 \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.

```

1736 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1737 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1738 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1739 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1740 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1741 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1742 \tl_gclear:N \g_nicematrix_code_before_tl
1743 \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]`).

```

1744 \dim_zero_new:N \l_@@_left_delim_dim
1745 \dim_zero_new:N \l_@@_right_delim_dim
1746 \bool_if:NTF \g_@@_delims_bool
1747 {

```

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

```

1748 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1749 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1750 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1751 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1752 }
1753 {
1754 \dim_gset:Nn \l_@@_left_delim_dim
1755 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1756 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1757 }
1758 }

```

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.

```

1759 \cs_new_protected:Npn \@@_pre_array:
1760 {
1761 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1762 \int_gzero_new:N \c@iRow
1763 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1764 \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).

```

1765 \int_compare:nNnT \l_@@_last_row_int > 0
1766 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1767 \int_compare:nNnT \l_@@_last_col_int > 0
1768 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1769 \bool_if:NT \g_@@_aux_found_bool
1770 {
1771 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }

```

```

1772     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1773     \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1774     \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1775 }

```

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).

```

1776     \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1777     {
1778         \bool_set_true:N \l_@@_last_row_without_value_bool
1779         \bool_if:NT \g_@@_aux_found_bool
1780             { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1781     }
1782     \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1783     {
1784         \bool_if:NT \g_@@_aux_found_bool
1785             { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1786     }

```

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”.

```

1787     \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1788     {
1789         \tl_put_right:Nn \@@_update_for_first_and_last_row:
1790         {
1791             \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1792             { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1793             \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1794             { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1795         }
1796     }

1797     \seq_gclear:N \g_@@_cols_vlism_seq
1798     \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1799     \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1800     \@@_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).

```

1801     \hbox_set:Nw \l_@@_the_array_box
1802     \skip_horizontal:N \l_@@_left_margin_dim
1803     \skip_horizontal:N \l_@@_extra_left_margin_dim
1804     \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).

```

1805     \m@th
1806     $ % $
1807     \bool_if:NTF \l_@@_light_syntax_bool
1808         { \use:c { @@-light-syntax } }
1809         { \use:c { @@-normal-syntax } }
1810 }

```

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

```

1811 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1812 {
1813   \tl_set:Nn \l_tmpa_tl { #1 }
1814   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1815     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1816   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1817   \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...

```

1818   \@@_pre_array:
1819 }

```

## 9 The `\CodeBefore`

```

1820 \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).

```

1821 \cs_new_protected:Npn \@@_pre_code_before:
1822 {

```

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.

```

1823   \pgfsys@markposition { \@@_env: - position }
1824   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1825   \pgfpicture
1826   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1827   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1828   {
1829     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1830     \pgfcoordinate { \@@_env: - row - ##1 }
1831     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1832   }

```

Now, the recreation of the `col` nodes.

```

1833   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1834   {
1835     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1836     \pgfcoordinate { \@@_env: - col - ##1 }
1837     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1838   }

```

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

```

1839   \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1840   \endpgfpicture

```

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

```

1841   \@@_create_diag_nodes:

```

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

```

1842   \@@_create_blocks_nodes:
1843   \IfPackageLoadedT { tikz }
1844   {
1845     \tikzset
1846     {

```

```

1847         every-picture / .style =
1848         { overlay , name~prefix = \@@_env: - }
1849     }
1850 }
1851 \cs_set_eq:NN \cellcolor \@@_cellcolor
1852 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1853 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1854 \cs_set_eq:NN \rowcolor \@@_rowcolor
1855 \cs_set_eq:NN \rowcolors \@@_rowcolors
1856 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1857 \cs_set_eq:NN \arraycolor \@@_arraycolor
1858 \cs_set_eq:NN \columncolor \@@_columncolor
1859 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1860 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1861 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1862 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1863 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1864 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1865 }

```

```

1866 \cs_new_protected:Npn \@@_exec_code_before:
1867 {

```

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...

```

1868 \clist_map_inline:Nn \l_@@_corners_cells_clist
1869 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1870 \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`.

```

1871 \@@_add_to_colors_seq:nn { { nocolor } } { }
1872 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1873 \group_begin:

```

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

```

1874 \if_mode_math:
1875 \@@_exec_code_before_i:
1876 \else:
1877 $ % $
1878 \@@_exec_code_before_i:
1879 $ % $
1880 \fi:
1881 \group_end:
1882 }

```

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`).

```

1883 \cs_new_protected:Npn \@@_exec_code_before_i:
1884 {
1885 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1886 { \@@_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`.

```

1887 \exp_last_unbraced:No \@@_CodeBefore_keys:
1888 \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.

```

1889     \@@_actually_color:
1890     \l_@@_code_before_tl
1891     \q_stop
1892 }

1893 \keys_define:nn { nicematrix / CodeBefore }
1894 {
1895     create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1896     create-cell-nodes .default:n = true ,
1897     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1898     sub-matrix .value_required:n = true ,
1899     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1900     delimiters / color .value_required:n = true ,
1901     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1902 }

1903 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1904 {
1905     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1906     \@@_CodeBefore:w
1907 }

```

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.

```

1908 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1909 {
1910     \bool_if:NTF \g_@@_aux_found_bool
1911     {
1912         \@@_pre_code_before:
1913         \legacy_if:nF { measuring@ } { #1 }
1914     }

```

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).

```

1915     {
1916         \pgfsys@markposition { \@@_env: - position }
1917         \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1918         \pgfpicture
1919         \pgf@relevantforpicturesizefalse
1920         \endpgfpicture

```

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

```

1921         \pgfpicture
1922         \pgfrememberpicturepositiononpagetrue
1923         \endpgfpicture

```

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

```

1924         \pgfpicture
1925         \pgf@relevantforpicturesizefalse
1926         \pgfrememberpicturepositiononpagetrue
1927         \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1928         \pgfpicture
1929         \pgf@relevantforpicturesizefalse
1930         \endpgfpicture
1931     }
1932 }

```

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.

```

1933 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1934 {
1935   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1936   {
1937     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1938     \pgfcoordinate { \@@_env: - row - ##1 - base }
1939     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1940     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1941     {
1942       \cs_if_exist:cT
1943       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1944       {
1945         \pgfsys@getposition
1946         { \@@_env: - ##1 - #####1 - NW }
1947         \@@_node_position:
1948         \pgfsys@getposition
1949         { \@@_env: - ##1 - #####1 - SE }
1950         \@@_node_position_i:
1951         \@@_pgf_rect_node:nnn
1952         { \@@_env: - ##1 - #####1 }
1953         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1954         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1955       }
1956     }
1957   }
1958   \@@_create_extra_nodes:
1959   \@@_create_aliases_last:
1960 }

1961 \cs_new_protected:Npn \@@_create_aliases_last:
1962 {
1963   \int_step_inline:nn { \c@iRow }
1964   {
1965     \pgfnodealias
1966     { \@@_env: - ##1 - last }
1967     { \@@_env: - ##1 - \int_use:N \c@jCol }
1968   }
1969   \int_step_inline:nn { \c@jCol }
1970   {
1971     \pgfnodealias
1972     { \@@_env: - last - ##1 }
1973     { \@@_env: - \int_use:N \c@iRow - ##1 }
1974   }
1975   \pgfnodealias
1976   { \@@_env: - last - last }
1977   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1978 }

1979 \cs_new_protected:Npn \@@_create_blocks_nodes:
1980 {
1981   \pgfpicture
1982   \pgf@relevantforpicturesizefalse
1983   \pgfrememberpicturepositiononpagetrue
1984   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1985   { \@@_create_one_block_node:nnnnn ##1 }
1986   \endpgfpicture
1987 }

```

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>

```

1988 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1989 {
1990   \tl_if_empty:nF { #5 }
1991   {
1992     \@@_qpoint:n { col - #2 }
1993     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1994     \@@_qpoint:n { #1 }
1995     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1996     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1997     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1998     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1999     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
2000     \@@_pgf_rect_node:nnnnn
2001     { \@@_env: - #5 }
2002     { \dim_use:N \l_tmpa_dim }
2003     { \dim_use:N \l_tmpb_dim }
2004     { \dim_use:N \l_@@_tmpc_dim }
2005     { \dim_use:N \l_@@_tmpd_dim }
2006   }
2007 }

2008 \cs_new_protected:Npn \@@_patch_for_revtext:
2009 {
2010   \cs_set_eq:NN \@addamp \@addamp@LaTeX
2011   \cs_set_eq:NN \@array \@array@array
2012   \cs_set_eq:NN \@tabular \@tabular@array
2013   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
2014   \cs_set_eq:NN \array \array@array
2015   \cs_set_eq:NN \endarray \endarray@array
2016   \cs_set:Npn \endtabular { \endarray $\egroup } % $
2017   \cs_set_eq:NN \@mkpream \@mkpream@array
2018   \cs_set_eq:NN \@classx \@classx@array
2019   \cs_set_eq:NN \insert@column \insert@column@array
2020   \cs_set_eq:NN \@arraycr \@arraycr@array
2021   \cs_set_eq:NN \@xarraycr \@xarraycr@array
2022   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
2023 }

```

## 10 The environment {NiceArrayWithDelims}

```

2024 \NewDocumentEnvironment { NiceArrayWithDelims }
2025 { m m 0 { } m ! 0 { } t \CodeBefore }
2026 {
2027   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
2028   \@@_provide_pgfsyspdfmark:
2029   \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.

```

2030   \bgroup

2031   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2032   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2033   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2034   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

```

<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).

```

2035 \int_gzero:N \g_@@_block_box_int
2036 \dim_gzero:N \g_@@_width_last_col_dim
2037 \dim_gzero:N \g_@@_width_first_col_dim
2038 \bool_gset_false:N \g_@@_row_of_col_done_bool
2039 \str_if_empty:NT \g_@@_name_env_str
2040 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2041 \bool_if:NTF \l_@@_tabular_bool
2042 { \mode_leave_vertical: }
2043 { \@@_test_if_math_mode: }
2044 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2045 \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.

```

2046 \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.

```

2047 \cs_if_exist:NT \tikz@library@external@loaded
2048 {
2049   \tikzexternaldisable
2050   \cs_if_exist:NT \ifstandalone
2051     { \tikzset { external / optimize = false } }
2052 }

```

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

```

2053 \int_gincr:N \g_@@_env_int
2054 \bool_if:NF \l_@@_block_auto_columns_width_bool
2055 { \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.

```

2056 \seq_gclear:N \g_@@_blocks_seq
2057 \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`.

```

2058 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2059 \seq_gclear:N \g_@@_pos_of_xdots_seq
2060 \tl_gclear_new:N \g_@@_code_before_tl
2061 \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.

```

2062 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2063 {
2064   \bool_gset_true:N \g_@@_aux_found_bool
2065   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2066 }
2067 { \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.

```

2068 \tl_gclear:N \g_@@_aux_tl
2069 \tl_if_empty:NF \g_@@_code_before_tl
2070 {
2071   \bool_set_true:N \l_@@_code_before_bool
2072   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl

```

---

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

```

2073     }
2074     \tl_if_empty:NF \g_@@_pre_code_before_tl
2075     { \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`.

```

2076     \bool_if:NTF \g_@@_delims_bool
2077     { \keys_set:nn { nicematrix / pNiceArray } }
2078     { \keys_set:nn { nicematrix / NiceArray } }
2079     { #3 , #5 }

2080     \@@_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:.`

```

2081     \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2082 }

```

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

```

2083 {
2084   \bool_if:NTF \l_@@_light_syntax_bool
2085   { \use:c { end @@-light-syntax } }
2086   { \use:c { end @@-normal-syntax } }
2087   $ % $
2088   \skip_horizontal:N \l_@@_right_margin_dim
2089   \skip_horizontal:N \l_@@_extra_right_margin_dim
2090   \hbox_set_end:
2091   \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.

```

2092     \bool_if:NT \l_@@_width_used_bool
2093     {
2094       \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2095       { \@@_error_or_warning:n { width~without~X~columns } }
2096     }

```

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 `\l_@@_X_columns_dim` multiplied by  $x$ .

```

2097     \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2098     { \@@_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).

```

2099     \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2100     {
2101       \bool_if:NF \l_@@_last_row_without_value_bool
2102       {
2103         \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2104         {
2105           \@@_error:n { Wrong~last~row }
2106           \int_set_eq:NN \l_@@_last_row_int \c@iRow
2107         }
2108       }
2109     }

```

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>

```

2110 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2111 \bool_if:NTF \g_@@_last_col_found_bool
2112 { \int_gdecr:N \c@jCol }
2113 {
2114   \int_compare:nNtT { \l_@@_last_col_int } > { -1 }
2115   { \@@_error:n { last~col~not~used } }
2116 }

```

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

```

2117 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2118 \int_compare:nNtT { \l_@@_last_row_int } > { -1 }
2119 { \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).

```

2120 \int_if_zero:nT { \l_@@_first_col_int }
2121 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2122 \bool_if:nTF { ! \g_@@_delims_bool }
2123 {
2124   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2125   { \@@_use_arraybox_with_notes_c: }
2126   {
2127     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2128     { \@@_use_arraybox_with_notes_b: }
2129     { \@@_use_arraybox_with_notes: }
2130   }
2131 }

```

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).

```

2132 {
2133   \int_if_zero:nTF { \l_@@_first_row_int }
2134   {
2135     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2136     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2137   }
2138   { \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>

```

2139 \int_compare:nNtTF { \l_@@_last_row_int } > { -2 }
2140 {
2141   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2142   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2143 }
2144 { \dim_zero:N \l_tmpb_dim }
2145 \hbox_set:Nn \l_tmpa_box
2146 {
2147   \m@th
2148   $ % $
2149   \@@_color:o \l_@@_delimiters_color_tl
2150   \exp_after:wN \left \g_@@_left_delim_tl
2151   \vcenter
2152   {

```

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

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

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.

```

2153         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2154         \hbox
2155         {
2156             \bool_if:NTF \l_@@_tabular_bool
2157             { \skip_horizontal:n { - \tabcolsep } }
2158             { \skip_horizontal:n { - \arraycolsep } }
2159             \@@_use_arraybox_with_notes_c:
2160             \bool_if:NTF \l_@@_tabular_bool
2161             { \skip_horizontal:n { - \tabcolsep } }
2162             { \skip_horizontal:n { - \arraycolsep } }
2163         }

```

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

```

2164         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2165     }
2166     \exp_after:wN \right \g_@@_right_delim_tl
2167     $ % $
2168 }

```

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.

```

2169     \bool_if:NTF \l_@@_delimiters_max_width_bool
2170     {
2171         \@@_put_box_in_flow_bis:nn
2172         { \g_@@_left_delim_tl }
2173         { \g_@@_right_delim_tl }
2174     }
2175     \@@_put_box_in_flow:
2176 }

```

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. 95).

```

2177     \bool_if:NT \g_@@_last_col_found_bool
2178     { \skip_horizontal:N \g_@@_width_last_col_dim }
2179     \bool_if:NT \l_@@_preamble_bool
2180     {
2181         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2182         { \@@_err_columns_not_used: }
2183     }
2184     \@@_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.

```

2185     \egroup

```

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

```

2186     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2187     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2188     \iow_now:Ne \@mainaux
2189     {
2190         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2191         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2192         { \exp_not:o \g_@@_aux_tl }
2193     }
2194     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2195     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2196 }

```

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

```

2197 \cs_new_protected:Npn \@@_err_columns_not_used:

```

```

2198 {
2199   \@@_warning:n { columns-not-used }
2200   \cs_gset:Npn \@@_err_columns_not_used: { }
2201 }

```

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  $x$ .

```

2202 \cs_new_protected:Npn \@@_compute_width_X:
2203 {
2204   \tl_gput_right:Ne \g_@@_aux_tl
2205   {
2206     \bool_set_true:N \l_@@_X_columns_aux_bool
2207     \dim_set:Nn \l_@@_X_columns_dim
2208     {

```

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`

```

2209       \bool_lazy_and:nnTF
2210       { \g_@@_V_of_X_bool }
2211       { \l_@@_X_columns_aux_bool }
2212       { \dim_use:N \l_@@_X_columns_dim }
2213       {
2214         \dim_compare:nNnTF
2215         {
2216           \dim_abs:n
2217           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2218         }
2219         <
2220         { 0.001 pt }
2221         { \dim_use:N \l_@@_X_columns_dim }
2222         {
2223           \dim_eval:n
2224           {
2225             \l_@@_X_columns_dim
2226             +
2227             \fp_to_dim:n
2228             {
2229               (
2230                 \dim_eval:n
2231                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2232               )
2233               / \fp_use:N \g_@@_total_X_weight_fp
2234             }
2235           }
2236         }
2237       }
2238     }
2239   }
2240 }

```

## 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`.

```

2241 \cs_new_protected:Npn \@@_transform_preamble:
2242 {
2243   \@@_transform_preamble_i:
2244   \@@_transform_preamble_ii:
2245 }
2246 \cs_new_protected:Npn \@@_transform_preamble_i:
2247 {
2248   \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`).

```

2249   \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.

```

2250   \bool_gset_false:N \g_tmpb_bool

```

The following sequence will store the arguments of the successive `>` in the preamble.

```

2251   \tl_gclear_new:N \g_@@_pre_cell_tl

```

The counter `\l_tmpa_int` will count the number of consecutive occurrences of the symbol `|`.

```

2252   \int_zero:N \l_tmpa_int
2253   \tl_gclear:N \g_@@_array_preamble_tl
2254   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2255   {
2256     \tl_gset:Nn \g_@@_array_preamble_tl
2257     { ! { \skip_horizontal:N \arrayrulewidth } }
2258   }
2259   {
2260     \clist_if_in:NnT \l_@@_vlines_clist 1
2261     {
2262       \tl_gset:Nn \g_@@_array_preamble_tl
2263       { ! { \skip_horizontal:N \arrayrulewidth } }
2264     }
2265   }

```

Now, we actually make the preamble (which will be given to `{array}`). It will be stored in `\g_@@_array_preamble_tl`.

```

2266   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2267   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2268
2269   \@@_replace_columncolor:

```

```

2270 \cs_new_protected:Npn \@@_transform_preamble_ii:
2271 {

```

If there were delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```

2272   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2273   {
2274     \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2275     { \bool_gset_true:N \g_@@_delims_bool }
2276   }
2277   { \bool_gset_true:N \g_@@_delims_bool }

```

We want to remind whether there is a specifier `|` at the end of the preamble.

```

2278   \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).

```

2279 \int_if_zero:nTF { \l_@@_first_col_int }
2280 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2281 {
2282   \bool_if:NF \g_@@_delims_bool
2283   {
2284     \bool_if:NF \l_@@_tabular_bool
2285     {
2286       \clist_if_empty:NT \l_@@_vlines_clist
2287       {
2288         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2289         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2290       }
2291     }
2292   }
2293 }
2294 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2295 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2296 {
2297   \bool_if:NF \g_@@_delims_bool
2298   {
2299     \bool_if:NF \l_@@_tabular_bool
2300     {
2301       \clist_if_empty:NT \l_@@_vlines_clist
2302       {
2303         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2304         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2305       }
2306     }
2307   }
2308 }

```

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.

```

2309 \tag_if_active:F
2310 {

```

Moreover, when {NiceTabular\*} is used, the mechanism can’t be used for technical reasons. We test that situation with \l\_@@\_tabular\_width\_dim.

```

2311 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2312 {
2313   \tl_gput_right:Nn \g_@@_array_preamble_tl
2314   { > { \@@_err_too_many_cols: } 1 }
2315 }
2316 }
2317 }

```

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.

```

2318 \cs_new_protected:Npn \@@_rec_preamble:n #1
2319 {

```

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>

```
2320 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2321 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2322 {
```

Now, the columns defined by `\newcolumnntype` of array.

```
2323 \cs_if_exist:cTF { NC @ find @ #1 }
2324 {
2325 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2326 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2327 }
2328 {
2329 \str_if_eq:nnTF { #1 } { S }
2330 { \@@_fatal:n { unknown~column~type~S } }
2331 { \@@_fatal:nn { unknown~column~type } { #1 } }
2332 }
2333 }
2334 }
```

For `c`, `l` and `r`

```
2335 \cs_new_protected:Npn \@@_c: #1
2336 {
2337 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2338 \tl_gclear:N \g_@@_pre_cell_tl
2339 \tl_gput_right:Nn \g_@@_array_preamble_tl
2340 { > \@@_cell_begin: c < \@@_cell_end: }
```

We increment the counter of columns and then we test for the presence of a `<`.

```
2341 \int_gincr:N \c@jCol
2342 \@@_rec_preamble_after_col:n
2343 }

2344 \cs_new_protected:Npn \@@_l: #1
2345 {
2346 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2347 \tl_gclear:N \g_@@_pre_cell_tl
2348 \tl_gput_right:Nn \g_@@_array_preamble_tl
2349 {
2350 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2351 l
2352 < \@@_cell_end:
2353 }
2354 \int_gincr:N \c@jCol
2355 \@@_rec_preamble_after_col:n
2356 }

2357 \cs_new_protected:Npn \@@_r: #1
2358 {
2359 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2360 \tl_gclear:N \g_@@_pre_cell_tl
2361 \tl_gput_right:Nn \g_@@_array_preamble_tl
2362 {
2363 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2364 r
2365 < \@@_cell_end:
2366 }
2367 \int_gincr:N \c@jCol
2368 \@@_rec_preamble_after_col:n
2369 }
```

---

<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`.

For ! and @

```

2370 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2371 {
2372   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2373   \@@_rec_preamble:n
2374 }
2375 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2376 \cs_new_protected:cpn { @@ _ | : } #1
2377 {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

2378   \int_incr:N \l_tmpa_int
2379   \@@_make_preamble_i_i:n
2380 }

2381 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2382 {

```

Here, we can't use \str\_if\_eq:eeTF.

```

2383   \str_if_eq:nnTF { #1 } { | }
2384   { \use:c { @@ _ | : } | }
2385   { \@@_make_preamble_i_ii:nn { } #1 }
2386 }

```

The following constructions aims to allow cumulative blocks of options between square brackets such as in |[color=blue][tikz=dashed].

```

2387 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2388 {
2389   \str_if_eq:nnTF { #2 } { [ ] }
2390   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2391   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2392 }
2393 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2394 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2395 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2396 {
2397   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2398   \tl_gput_right:Ne \g_@@_array_preamble_tl
2399   {

```

Here, the command \dim\_use:N is mandatory.

```

2400   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2401 }
2402 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2403 {
2404   \@@_vline:n
2405   {
2406     position = \int_eval:n { \c@jCol + 1 } ,
2407     multiplicity = \int_use:N \l_tmpa_int ,
2408     total-width = \dim_use:N \l_@@_rule_width_dim ,
2409     #2
2410   }

```

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.

```

2411   }
2412   \int_zero:N \l_tmpa_int
2413   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2414   \@@_rec_preamble:n #1
2415 }

```

```

2416 \cs_new_protected:cpn { @@ _ > : } #1 #2
2417 {
2418   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2419   \@@_rec_preamble:n
2420 }
2421 \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.

```

2422 \keys_define:nn { nicematrix / p-column }
2423 {
2424   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2425   r .value_forbidden:n = true ,
2426   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2427   c .value_forbidden:n = true ,
2428   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2429   l .value_forbidden:n = true ,
2430   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2431   S .value_forbidden:n = true ,
2432   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2433   p .value_forbidden:n = true ,
2434   t .meta:n = p ,
2435   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2436   m .value_forbidden:n = true ,
2437   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2438   b .value_forbidden:n = true
2439 }

```

For `p` but also `b` and `m`.

```

2440 \cs_new_protected:Npn \@@_p: #1
2441 {
2442   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

Now, you look for a potential character `[` after the letter of the specifier (for the options).

```

2443   \@@_make_preamble_ii_i:n
2444 }
2445 \cs_set_eq:NN \@@_b: \@@_p:
2446 \cs_set_eq:NN \@@_m: \@@_p:
2447 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2448 {
2449   \str_if_eq:nnTF { #1 } { [ ]
2450     { \@@_make_preamble_ii_ii:w [ ]
2451       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2452     }
2453   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2454   { \@@_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.

```

2455 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2456 {

```

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).

```

2457   \str_set:Nn \l_@@_hpos_col_str { j }
2458   \@@_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`.

```

2459   \setlength { \l_tmpa_dim } { #2 }

```

```

2460 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2461 }
2462 \cs_new_protected:Npn \@@_keys_p_column:n #1
2463 { \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.

```

2464 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2465 {

```

Here, `\expanded` would probably be slightly faster than `\use:e`

```

2466 \use:e
2467 {
2468 \@@_make_preamble_ii_vi:nnnnnnnn
2469 { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2470 { #1 }
2471 {

```

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.

```

2472 \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2473 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2474 {

```

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

```

2475 \def \exp_not:N \l_@@_hpos_cell_tl
2476 { \str_lowercase:f { \l_@@_hpos_col_str } }
2477 }
2478 \IfPackageLoadedTF { ragged2e }
2479 {
2480 \str_case:on \l_@@_hpos_col_str
2481 {

```

The following `\exp_not:N` are mandatory.

```

2482 c { \exp_not:N \Centering }
2483 l { \exp_not:N \RaggedRight }
2484 r { \exp_not:N \RaggedLeft }
2485 }
2486 }
2487 {
2488 \str_case:on \l_@@_hpos_col_str
2489 {
2490 c { \exp_not:N \centering }
2491 l { \exp_not:N \raggedright }
2492 r { \exp_not:N \raggedleft }
2493 }
2494 }
2495 #3
2496 }
2497 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2498 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2499 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2500 { #2 }
2501 {
2502 \str_case:onF \l_@@_hpos_col_str
2503 {
2504 { j } { c }
2505 { si } { c }
2506 }

```

We use `\str_lowercase:n` to convert R to r, etc.

```

2507         { \str_lowercase:f \l_@@_hpos_col_str }
2508     }
2509 }

```

We increment the counter of columns, and then we test for the presence of a `<`.

```

2510     \int_gincr:N \c@jCol
2511     \@@_rec_preamble_after_col:n
2512 }

```

#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*.

```

2513 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2514 {
2515     \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2516     {
2517         \tl_gput_right:Nn \g_@@_array_preamble_tl
2518         { > \@@_test_if_empty_for_S: }
2519     }
2520     {
2521         \str_if_eq:eeTF { #7 } { varwidth }
2522         {
2523             \tl_gput_right:Nn \g_@@_array_preamble_tl
2524             { > \@@_test_if_empty_varwidth: }
2525         }
2526         { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2527     }
2528     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2529     \tl_gclear:N \g_@@_pre_cell_tl
2530     \tl_gput_right:Nn \g_@@_array_preamble_tl
2531     {
2532         > {

```

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.

```

2533         \dim_set:Nn \l_@@_col_width_dim { #2 }
2534         \@@_cell_begin:

```

We use the form `\minipage–\endminipage` (`\varwidth–\endvarwidth`) for compatibility with `colcell` (2023-10-31).

```

2535         \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty`.

```

2536     \everypar
2537     {
2538         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2539         \everypar { }
2540     }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\RaggedRight`, etc.).

```

2541     #3

```

The following code is to allow something like `\centering` in `\RowStyle`.

```

2542         \g_@@_row_style_tl
2543         \arraybackslash
2544         #5
2545     }
2546     #8
2547     < {
2548         #6

```

The following line has been taken from `array.sty`.

```

2549         \@finalstrut \@arstrutbox
2550         \use:c { end #7 }

```

If the letter in the preamble is `m`, `#4` will be equal to `\@@_center_cell_box:` (see just below).

```

2551         #4
2552         \@@_cell_end:
2553     }
2554 }
2555 }

```

The cell always begins with `\ignorespaces` with `array` and that's why we retrieve that token.

```

2556 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2557 {

```

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 `&`).

```

2558     \group_align_safe_begin:
2559     \peek_meaning:NTF &
2560     { \@@_the_cell_is_empty: }
2561     {
2562         \peek_meaning:NTF \\\
2563         { \@@_the_cell_is_empty: }
2564         {
2565             \peek_meaning:NTF \crcr
2566             \@@_the_cell_is_empty:
2567             \group_align_safe_end:
2568         }
2569     }
2570 }

```

A special version of the previous function for the columns of type `V` (of `varwidth`).

```

2571 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2572 {
2573     \group_align_safe_begin:
2574     \peek_meaning:NTF &
2575     { \@@_the_cell_is_empty_varwidth: }
2576     {
2577         \peek_meaning:NTF \\\
2578         { \@@_the_cell_is_empty_varwidth: }
2579         {
2580             \peek_meaning:NTF \crcr
2581             \@@_the_cell_is_empty_varwidth:
2582             \group_align_safe_end:
2583         }
2584     }
2585 }

2586 \cs_new_protected:Npn \@@_the_cell_is_empty:
2587 {
2588     \group_align_safe_end:
2589     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2590     {

```

Be careful: here, we can't merely use `\bool_gset_true: \g_@@_empty_cell_bool`, in particular because of the columns of type X.

```
2591 \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).

```
2592 \skip_horizontal:N \l_@@_col_width_dim
```

```
2593 }
```

```
2594 }
```

```
2595 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
```

```
2596 {
```

```
2597 \group_align_safe_end:
```

```
2598 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2599 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
```

```
2600 }
```

```
2601 \cs_new_protected:Npn \@@_test_if_empty_for_S:
```

```
2602 {
```

```
2603 \peek_meaning:NT \__siunitx_table_skip:n
```

```
2604 { \bool_gset_true:N \g_@@_empty_cell_bool }
```

```
2605 }
```

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 that the height of `\strutbox`, there is only one row.

```
2606 \cs_new_protected:Npn \@@_center_cell_box:
```

```
2607 {
```

By putting instructions in `\g_@@_cell_after_hook_tl`, we require a post-action of the box `\l_@@_cell_box`.

```
2608 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
```

```
2609 {
```

```
2610 \dim_compare:nNnT
```

```
2611 { \box_ht:N \l_@@_cell_box }
```

```
2612 >
```

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).

```
2613 { \box_ht:N \strutbox }
```

```
2614 {
```

```
2615 \hbox_set:Nn \l_@@_cell_box
```

```
2616 {
```

```
2617 \box_move_down:nn
```

```
2618 {
```

```
2619 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
```

```
2620 + \baselineskip ) / 2
```

```
2621 }
```

```
2622 { \box_use:N \l_@@_cell_box }
```

```
2623 }
```

```
2624 }
```

```
2625 }
```

```
2626 }
```

For V (similar to the V of `varwidth`).

```
2627 \cs_new_protected:Npn \@@_V: #1 #2
```

```
2628 {
```

```
2629 \str_if_eq:nnTF { #2 } { [ ] }
```

```
2630 { \@@_make_preamble_V_i:w [ ] }
```

```
2631 { \@@_make_preamble_V_i:w [ ] { #2 } }
```

```
2632 }
```

```

2633 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2634 { \@@_make_preamble_V_ii:nn { #1 } }
2635 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2636 {
2637   \str_set:Nn \l_@@_vpos_col_str { p }
2638   \str_set:Nn \l_@@_hpos_col_str { j }
2639   \@@_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`.

```

2640 \setlength { \l_tmpa_dim } { #2 }
2641 \IfPackageLoadedTF { varwidth }
2642 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2643 {
2644   \@@_error_or_warning:n { varwidth-not-loaded }
2645   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2646 }
2647 }

```

For `w` and `W`

```

2648 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2649 \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.

```

2650 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2651 {
2652   \str_if_eq:nnTF { #3 } { s }
2653   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2654   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2655 }

```

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.

```

2656 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2657 {
2658   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2659   \tl_gclear:N \g_@@_pre_cell_tl
2660   \tl_gput_right:Nn \g_@@_array_preamble_tl
2661   {
2662     > {

```

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`.

```

2663 \setlength { \l_@@_col_width_dim } { #2 }
2664 \@@_cell_begin:
2665 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2666 }
2667 c
2668 < {
2669   \@@_cell_end_for_w_s:
2670   #1
2671   \@@_adjust_size_box:
2672   \box_use_drop:N \l_@@_cell_box
2673 }
2674 }
2675 \int_gincr:N \c@jCol

```



```

2676 \@@_rec_preamble_after_col:n
2677 }

```

Then, the most important version, for the horizontal alignments types of c, l and r (and not s).

```

2678 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2679 {
2680   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2681   \tl_gclear:N \g_@@_pre_cell_tl
2682   \tl_gput_right:Nn \g_@@_array_preamble_tl
2683   {
2684     > {

```

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`.

```

2685   \setlength { \l_@@_col_width_dim } { #4 }
2686   \hbox_set:Nw \l_@@_cell_box
2687   \@@_cell_begin:
2688   \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2689 }
2690 c
2691 < {
2692   \@@_cell_end:
2693   \hbox_set_end:
2694   #1
2695   \@@_adjust_size_box:
2696   \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2697 }
2698 }

```

We increment the counter of columns and then we test for the presence of a <.

```

2699 \int_gincr:N \c@jCol
2700 \@@_rec_preamble_after_col:n
2701 }

```

```

2702 \cs_new_protected:Npn \@@_special_W:
2703 {
2704   \dim_compare:nnTF { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2705   { \@@_warning:n { W-warning } }
2706 }

```

For S (of siunitx).

```

2707 \cs_new_protected:Npn \@@_S: #1 #2
2708 {
2709   \str_if_eq:nnTF { #2 } { [ ] }
2710   { \@@_make_preamble_S:w [ ] }
2711   { \@@_make_preamble_S:w [ ] { #2 } }
2712 }
2713 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2714 { \@@_make_preamble_S:i:n { #1 } }
2715 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2716 {
2717   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2718   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2719   \tl_gclear:N \g_@@_pre_cell_tl
2720   \tl_gput_right:Nn \g_@@_array_preamble_tl
2721   {
2722     > {

```

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).

```

2723         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2724         \keys_set:nn { siunitx } { #1 }
2725         \@@_cell_begin:
2726         \siunitx_cell_begin:w
2727     }
2728     c
2729     <
2730     {
2731         \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`).

```

2732         \tl_gput_right:Nc \g_@@_cell_after_hook_tl
2733         {
2734             \bool_if:NTF \l__siunitx_table_text_bool
2735             { \bool_set_true:N }
2736             { \bool_set_false:N }
2737             \l__siunitx_table_text_bool
2738         }
2739         \@@_cell_end:
2740     }
2741 }

```

We increment the counter of columns and then we test for the presence of a `<`.

```

2742     \int_gincr:N \c@jCol
2743     \@@_rec_preamble_after_col:n
2744 }

```

For `(`, `[` and `\{`.

```

2745 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2746 {
2747     \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.

```

2748     \int_if_zero:nTF { \c@jCol }
2749     {
2750         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2751         {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

2752         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2753         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2754         \@@_rec_preamble:n #2
2755     }
2756     {
2757         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2758         \@@_make_preamble_iv:nn { #1 } { #2 }
2759     }
2760 }
2761 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2762 }
2763 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2764 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2765 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2766 {
2767     \tl_gput_right:Nc \g_@@_pre_code_after_tl
2768     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2769 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2770 {
2771   \@@_error:nn { delimiter~after~opening } { #2 }
2772   \@@_rec_preamble:n
2773 }
2774 { \@@_rec_preamble:n #2 }
2775 }

```

In fact, it would be possible to define `\left` and `\right` as no-op.

```

2776 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2777 { \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}`).

```

2778 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2779 {
2780   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2781   \tl_if_in:nnTF { ) ] \} } { #2 }
2782   { \@@_make_preamble_v:nnn #1 #2 }
2783   {
2784     \str_if_eq:nnTF { \s_stop } { #2 }
2785     {
2786       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2787       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2788       {
2789         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2790         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2791         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2792         \@@_rec_preamble:n #2
2793       }
2794     }
2795     {
2796       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2797       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2798       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2799       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2800       \@@_rec_preamble:n #2
2801     }
2802   }
2803 }
2804 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2805 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2806 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2807 {
2808   \str_if_eq:nnTF { \s_stop } { #3 }
2809   {
2810     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2811     {
2812       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2813       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2814       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2815       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2816     }
2817     {
2818       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2819       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2820       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2821       \@@_error:nn { double~closing~delimiter } { #2 }
2822     }
2823   }
2824   {

```

```

2825 \tl_gput_right:Nn \g_@@_pre_code_after_tl
2826 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2827 \@@_error:nn { double~closing~delimiter } { #2 }
2828 \@@_rec_preamble:n #3
2829 }
2830 }

2831 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2832 { \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 @{...}.

```

2833 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2834 {
2835   \str_if_eq:nnTF { #1 } { < }
2836   { \@@_rec_preamble_after_col_i:n }
2837   {
2838     \str_if_eq:nnTF { #1 } { @ }
2839     { \@@_rec_preamble_after_col_ii:n }
2840     {
2841       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2842       {
2843         \tl_gput_right:Nn \g_@@_array_preamble_tl
2844         { ! { \skip_horizontal:N \arrayrulewidth } }
2845       }
2846       {
2847         \clist_if_in:Nt \l_@@_vlines_clist
2848         { \int_eval:n { \c@jCol + 1 } }
2849         {
2850           \tl_gput_right:Nn \g_@@_array_preamble_tl
2851           { ! { \skip_horizontal:N \arrayrulewidth } }
2852         }
2853       }
2854       \@@_rec_preamble:n { #1 }
2855     }
2856   }
2857 }

2858 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2859 {
2860   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2861   \@@_rec_preamble_after_col:n
2862 }

```

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.

```

2863 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2864 {
2865   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2866   {
2867     \tl_gput_right:Nn \g_@@_array_preamble_tl
2868     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2869   }
2870   {
2871     \clist_if_in:NtTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2872     {
2873       \tl_gput_right:Nn \g_@@_array_preamble_tl
2874       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2875     }
2876     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2877   }
2878   \@@_rec_preamble:n
2879 }

```

```

2880 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2881 {
2882   \tl_clear:N \l_tmpa_tl
2883   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2884   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2885 }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumntype`. We want that token to be no-op here.

```

2886 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2887 { \@@_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`.

```

2888 \cs_new_protected:Npn \@@_X: #1 #2
2889 {
2890   \str_if_eq:nnTF { #2 } { [ ]
2891     { \@@_make_preamble_X:w [ ] }
2892     { \@@_make_preamble_X:w [ ] #2 }
2893   }
2894   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2895   { \@@_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`.

```

2896 \keys_define:nn { nicematrix / X-column }
2897 {
2898   V .code:n =
2899     \IfPackageLoadedTF { varwidth }
2900     {
2901       \bool_set_true:N \l_@@_V_of_X_bool
2902       \bool_gset_true:N \g_@@_V_of_X_bool
2903     }
2904     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2905   unknown .code:n =
2906     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2907     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2908     { \@@_error_or_warning:n { invalid~weight } }
2909 }

```

In the following command, `#1` is the list of the options of the specifier `X`.

```

2910 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2911 {

```

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`).

```

2912   \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`).

```

2913   \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`).

```

2914   \fp_set:Nn \l_tmpa_fp { 1.0 }
2915   \@@_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.

```
2916 \bool_set_false:N \l_@@_V_of_X_bool
2917 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

Now, the weight of the column is stored in `\l_tmpa_tl`.

```
2918 \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).

```
2919 \bool_if:NTF \l_@@_X_columns_aux_bool
2920 {
2921 \@@_make_preamble_ii_iv:nnn
```

Of course, the weight of a column depends of its weight (in `\l_tmpa_fp`).

```
2922 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2923 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2924 { \@@_no_update_width: }
2925 }
```

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.

```
2926 {
2927 \tl_gput_right:Nn \g_@@_array_preamble_tl
2928 {
2929 > {
2930 \@@_cell_begin:
2931 \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`.

```
2932 \NotEmpty
```

The following code will nullify the box of the cell.

```
2933 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2934 { \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`.

```
2935 \begin { minipage } { 5 cm } \arraybackslash
2936 }
2937 c
2938 < {
2939 \end { minipage }
2940 \@@_cell_end:
2941 }
2942 }
2943 \int_gincr:N \c@jCol
2944 \@@_rec_preamble_after_col:n
2945 }
2946 }
```

```
2947 \cs_new_protected:Npn \@@_no_update_width:
2948 {
2949 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2950 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2951 }
```

For the letter set by the user with `vlines-in-sub-matrix` (`vlism`).

```

2952 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2953 {
2954   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2955   { \int_eval:n { \c@jCol + 1 } }
2956   \tl_gput_right:Ne \g_@@_array_preamble_tl
2957   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2958   \@@_rec_preamble:n
2959 }

```

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.

```

2960 \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).

```

2961 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2962 { \@@_fatal:n { Preamble-forgotten } }
2963 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2964 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2965 { @@ _ \token_to_str:N \hline : }
2966 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2967 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2968 { @@ _ \token_to_str:N \hline : }
2969 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2970 { @@ _ \token_to_str:N \hline : }
2971 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2972 { @@ _ \token_to_str:N \hline : }
2973 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2974 { @@ _ \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).

```

2975 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2976 {

```

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`.

```

2977   \multispan { #1 }
2978   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2979   \begingroup
2980   \tbl_update_multicolumn_cell_data:n { #1 }

```

Now, we patch the (small) preamble as we have done with the main preamble of the array.

```

2981   \tl_gclear:N \g_@@_preamble_tl
2982   \@@_make_m_preamble:n #2 \q_stop

```

The following lines are an adaptation of the definition of `\multicolumn` in `array`.

```

2983   \def \@addamp
2984   {
2985     \legacy_if:nTF { @firstamp }
2986     { \legacy_if_set_false:n { @firstamp } }
2987     { \@preamerr 5 }
2988   }
2989   \exp_args:No \@mkpream \g_@@_preamble_tl
2990   \@addtopreamble \@empty

```

```

2991 \endgroup
2992 \UseTaggingSocket { tbl / colspan } { #1 }

```

Now, we do a treatment specific to nicematrix which has no equivalent in the original definition of `\multicolumn`.

```

2993 \int_compare:nNtT { #1 } > { \c_one_int }
2994 {
2995   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2996   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2997   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2998   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2999   {
3000     {
3001       \int_if_zero:nTF { \c@jCol }
3002       { \int_eval:n { \c@iRow + 1 } }
3003       { \int_use:N \c@iRow }
3004     }
3005     { \int_eval:n { \c@jCol + 1 } }
3006     {
3007       \int_if_zero:nTF { \c@jCol }
3008       { \int_eval:n { \c@iRow + 1 } }
3009       { \int_use:N \c@iRow }
3010     }
3011     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3012   { }
3013 }
3014 }

```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of `colortbl` is available in `\multicolumn`.

```

3015 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3016 {
3017   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3018   {
3019     \@@_rectanglecolor [ ##1 ]
3020     { \exp_not:n { ##2 } }
3021     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3022     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3023   }
3024   \ignorespaces
3025 }

```

The following lines were in the original definition of `\multicolumn`.

```

3026 \def \@sharp { #3 }
3027 \@arstrut
3028 \@preamble
3029 \null

```

We add some lines.

```

3030 \int_gadd:Nn \c@jCol { #1 - 1 }
3031 \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
3032 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3033 \ignorespaces
3034 }

```

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`.

```

3035 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3036 {
3037   \str_case:nnF { #1 }

```



```

3038 {
3039   c { \@@_make_m_preamble_i:n #1 }
3040   l { \@@_make_m_preamble_i:n #1 }
3041   r { \@@_make_m_preamble_i:n #1 }
3042   > { \@@_make_m_preamble_ii:nn #1 }
3043   ! { \@@_make_m_preamble_ii:nn #1 }
3044   @ { \@@_make_m_preamble_ii:nn #1 }
3045   | { \@@_make_m_preamble_iii:n #1 }
3046   p { \@@_make_m_preamble_iv:nnn t #1 }
3047   m { \@@_make_m_preamble_iv:nnn c #1 }
3048   b { \@@_make_m_preamble_iv:nnn b #1 }
3049   w { \@@_make_m_preamble_v:nnnn { } #1 }
3050   W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3051   \q_stop { }
3052 }
3053 {
3054   \cs_if_exist:cTF { NC @ find @ #1 }
3055   {
3056     \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3057     \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3058   }
3059   {
3060     \str_if_eq:nnTF { #1 } { S }
3061     { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3062     { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3063   }
3064 }
3065 }

```

For c, l and r

```

3066 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3067 {
3068   \tl_gput_right:Nn \g_@@_preamble_tl
3069   {
3070     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3071     #1
3072     < \@@_cell_end:
3073   }

```

We test for the presence of a <.

```

3074   \@@_make_m_preamble_x:n
3075 }

```

For >, ! and @

```

3076 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3077 {
3078   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3079   \@@_make_m_preamble:n
3080 }

```

For |

```

3081 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3082 {
3083   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3084   \@@_make_m_preamble:n
3085 }

```

For p, m and b

```

3086 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3087 {
3088   \tl_gput_right:Nn \g_@@_preamble_tl
3089   {
3090     > {
3091       \@@_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`.

```

3092         \setlength { \l_tmpa_dim } { #3 }
3093         \begin { minipage } [ #1 ] { \l_tmpa_dim }
3094         \mode_leave_vertical:
3095         \arraybackslash
3096         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3097     }
3098     c
3099     < {
3100         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3101         \end { minipage }
3102         \@@_cell_end:
3103     }
3104 }
```

We test for the presence of a `<`.

```

3105     \@@_make_m_preamble_x:n
3106 }
```

For `w` and `W`

```

3107 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3108 {
3109     \tl_gput_right:Nn \g_@@_preamble_tl
3110     {
3111         > {
3112             \dim_set:Nn \l_@@_col_width_dim { #4 }
3113             \hbox_set:Nw \l_@@_cell_box
3114             \@@_cell_begin:
3115             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3116         }
3117         c
3118         < {
3119             \@@_cell_end:
3120             \hbox_set_end:
3121             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3122             #1
3123             \@@_adjust_size_box:
3124             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3125         }
3126     }
}
```

We test for the presence of a `<`.

```

3127     \@@_make_m_preamble_x:n
3128 }
```

After a specifier of column, we have to test whether there is one or several `<{...}`.

```

3129 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3130 {
3131     \str_if_eq:nnTF { #1 } { < }
3132     { \@@_make_m_preamble_ix:n }
3133     { \@@_make_m_preamble:n { #1 } }
3134 }
3135 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3136 {
3137     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3138     \@@_make_m_preamble_x:n
3139 }
```

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`).

```

3140 \cs_new_protected:Npn \@@_put_box_in_flow:
3141 {
3142   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3143   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3144   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3145     { \box_use_drop:N \l_tmpa_box }
3146     { \@@_put_box_in_flow_i: }
3147 }

```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different of `c` (the initial value).

```

3148 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3149 {
3150   \pgfpicture
3151     \@@_qpoint:n { row - 1 }
3152     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3153     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3154     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3155     \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).

```

3156   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3157   {
3158     \int_set:Nn \l_tmpa_int
3159       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3160     \bool_lazy_or:nnT
3161       { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3162       { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3163       {
3164         \@@_error:n { bad-value-for-baseline-line }
3165         \int_set_eq:NN \l_tmpa_int \c_one_int
3166       }
3167     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3168   }
3169   {
3170     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3171     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3172     {
3173       \str_if_eq:onTF \l_@@_baseline_tl { b }
3174       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3175       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3176     }
3177     \bool_lazy_or:nnT
3178       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3179       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3180       {
3181         \@@_error:n { bad-value-for-baseline }
3182         \int_set_eq:NN \l_tmpa_int \c_one_int
3183       }
3184     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3185     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3186   }
3187   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

Now, `\g_tmpa_dim` contains the value of the  $y$  translation we have to to.

```

3188   \endpgfpicture
3189   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3190   \box_use_drop:N \l_tmpa_box
3191 }

```

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).

```

3192 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3193 {

```

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.

```

3194 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3195 {
3196   \int_compare:nNt { \c@jCol } > { \c_one_int }
3197   {
3198     \box_set_wd:Nn \l_@@_the_array_box
3199     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3200   }
3201 }

```

We need a `{minipage}` because we will insert a LaTeX list for the tabular notes (that means that a `\vtop{\hsize=...}` is not enough).

```

3202 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3203 \bool_if:NT \l_@@_caption_above_bool
3204 {
3205   \tl_if_empty:NF \l_@@_caption_tl
3206   {
3207     \bool_set_false:N \g_@@_caption_finished_bool
3208     \int_gzero:N \c@tabularnote
3209     \@@_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).

```

3210   \int_compare:nNt { \g_@@_notes_caption_int } > { \c_zero_int }
3211   {
3212     \tl_gput_right:Ne \g_@@_aux_tl
3213     {
3214       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3215       { \int_use:N \g_@@_notes_caption_int }
3216     }
3217     \int_gzero:N \g_@@_notes_caption_int
3218   }
3219 }
3220 }

```

The `\hbox` avoids that the `pgfpicture` inside `\@@_draw_blocks` adds a extra vertical space before the notes.

```

3221 \hbox
3222 {
3223   \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.

```

3224 \@@_create_extra_nodes:
3225 \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3226 }

```

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).

```

3227 \bool_lazy_any:nT
3228 {
3229   { ! \seq_if_empty_p:N \g_@@_notes_seq }

```

```

3230     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3231     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3232   }
3233   \@@_insert_tabularnotes:
3234   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3235   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3236   \end { minipage }
3237 }

```

```

3238 \cs_new_protected:Npn \@@_insert_caption:
3239 {
3240   \tl_if_empty:NF \l_@@_caption_tl
3241   {
3242     \cs_if_exist:NTF \@capytype
3243     { \@@_insert_caption_i: }
3244     { \@@_error:n { caption~outside~float } }
3245   }
3246 }

```

```

3247 \cs_new_protected:Npn \@@_insert_caption_i:
3248 {
3249   \group_begin:

```

The flag `\l_@@_in_caption_bool` affects only the behavior of the command `\tabularnote` when used in the caption.

```

3250   \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.

```

3251   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3252   \tl_if_empty:NTF \l_@@_short_caption_tl
3253   { \caption }
3254   { \caption [ \l_@@_short_caption_tl ] }
3255   { \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.

```

3256   \bool_if:NF \g_@@_caption_finished_bool
3257   {
3258     \bool_gset_true:N \g_@@_caption_finished_bool
3259     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3260     \int_gzero:N \c@tabularnote
3261   }
3262   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3263   \group_end:
3264 }

```

```

3265 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3266 {
3267   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3268   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3269 }

```

```

3270 \cs_new_protected:Npn \@@_insert_tabularnotes:
3271 {
3272   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3273   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3274   \skip_vertical:N 0.65ex

```

The TeX group is for potential specifications in the `\l_@@_notes_code_before_tl`.

```

3275 \group_begin:
3276 \l_@@_notes_code_before_tl
3277 \tl_if_empty:NF \g_@@_tabularnote_tl
3278 {
3279   \g_@@_tabularnote_tl \par
3280   \tl_gclear:N \g_@@_tabularnote_tl
3281 }

```

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.

```

3282 \int_compare:nNtT { \c@tabularnote } > { \c_zero_int }
3283 {
3284   \bool_if:NTF \l_@@_notes_para_bool
3285   {
3286     \begin { tabularnotes* }
3287     \seq_map_inline:Nn \g_@@_notes_seq
3288       { \@@_one_tabularnote:nn ##1 }
3289     \strut
3290     \end { tabularnotes* }

```

The following `\par` is mandatory for the event that the user has put `\footnotesize` (for example) in the `notes/code-before`.

```

3291 \par
3292 }
3293 {
3294   \tabularnotes
3295   \seq_map_inline:Nn \g_@@_notes_seq
3296     { \@@_one_tabularnote:nn ##1 }
3297   \strut
3298   \endtabularnotes
3299 }
3300 }
3301 \unskip
3302 \group_end:
3303 \bool_if:NT \l_@@_notes_bottomrule_bool
3304 {
3305   \IfPackageLoadedTF { booktabs }
3306   {

```

The two dimensions `\aboverulesep` et `\heavyrulewidth` are parameters defined by `booktabs`.

```

3307 \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

3308 { \CT@arc@ \hrule height \heavyrulewidth }
3309 }
3310 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3311 }
3312 \l_@@_notes_code_after_tl
3313 \seq_gclear:N \g_@@_notes_seq
3314 \seq_gclear:N \g_@@_notes_in_caption_seq
3315 \int_gzero:N \c@tabularnote
3316 }

```

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 currying.

```

3317 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3318 {
3319   \tl_if_novalue:nTF { #1 }
3320   { \item }
3321   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3322 }

```

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`.

```

3323 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3324 {
3325   \pgfpicture
3326     \@@_qpoint:n { row - 1 }
3327     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3328     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3329     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3330   \endpgfpicture
3331   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3332   \int_if_zero:nT { \l_@@_first_row_int }
3333   {
3334     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3335     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3336   }
3337   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3338 }

```

Now, the general case.

```

3339 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3340 {

```

We convert a value of `t` to a value of 1.

```

3341   \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3342   { \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`.

```

3343   \pgfpicture
3344   \@@_qpoint:n { row - 1 }
3345   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3346   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3347   {
3348     \int_set:Nn \l_tmpa_int
3349     {
3350       \str_range:Nnn
3351         \l_@@_baseline_tl
3352         { 6 }
3353         { \tl_count:o \l_@@_baseline_tl }
3354     }
3355     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3356   }
3357   {
3358     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3359     \bool_lazy_or:nnT
3360       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3361       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3362     {
3363       \@@_error:n { bad-value-for-baseline }
3364       \int_set:Nn \l_tmpa_int 1
3365     }
3366     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3367   }
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 }
```

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.

```
3378 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3379 {
```

We will compute the real width of both delimiters used.

```
3380 \dim_zero_new:N \l_@@_real_left_delim_dim
3381 \dim_zero_new:N \l_@@_real_right_delim_dim
3382 \hbox_set:Nn \l_tmpb_box
3383 {
3384   \m@th
3385   $ % $
3386   \left #1
3387   \vcenter
3388   {
3389     \vbox_to_ht:nn
3390     { \box_ht_plus_dp:N \l_tmpa_box }
3391     { }
3392   }
3393   \right .
3394   $ % $
3395 }
3396 \dim_set:Nn \l_@@_real_left_delim_dim
3397 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3398 \hbox_set:Nn \l_tmpb_box
3399 {
3400   \m@th
3401   $ % $
3402   \left .
3403   \vbox_to_ht:nn
3404   { \box_ht_plus_dp:N \l_tmpa_box }
3405   { }
3406   \right #2
3407   $ % $
3408 }
3409 \dim_set:Nn \l_@@_real_right_delim_dim
3410 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.

```
3411 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3412 \@@_put_box_in_flow:
3413 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3414 }
```

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).

```
3415 \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).

```
3416 {
3417   \peek_remove_spaces:n
3418   {
3419     \peek_meaning:NTF \end
3420     { \@@_analyze_end:Nn }
```



```

3421     {
3422         \@@_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).
3423         \@@_array:o \g_@@_array_preamble_tl
3424     }
3425 }
3426 }
3427 {
3428     \@@_create_col_nodes:
3429     \endarray
3430 }

```

When the key `light-syntax` is in force, we use an environment which takes its whole body as an argument (with the specifier `b`).

```

3431 \NewDocumentEnvironment { @@-light-syntax } { b }
3432 {

```

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`.

```

3433     \tl_if_empty:nT { #1 }
3434     { \@@_fatal:n { empty-environment } }
3435     \tl_if_in:nnT { #1 } { & }
3436     { \@@_fatal:n { ampersand-in-light-syntax } }
3437     \tl_if_in:nnT { #1 } { \ }
3438     { \@@_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`.

```

3439     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

The command `\array` is hidden somewhere in `\@@_light_syntax_i:w`.

```

3440 }

```

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.

```

3441 {
3442     \@@_create_col_nodes:
3443     \endarray
3444 }
3445 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3446 {
3447     \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).

```

3448     \seq_clear_new:N \l_@@_rows_seq

```

We rescan the character of end of line in order to have the correct catcode.

```

3449     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3450     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3451     { \seq_set_split:Nee }
3452     { \seq_set_split:Non }
3453     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3454     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3455     \tl_if_empty:NF \l_tmpa_tl
3456     { \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.

```

3457 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3458 { \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`).

```

3459 \tl_build_begin:N \l_@@_new_body_tl
3460 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3461 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3462 \@@_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).

```

3463 \seq_map_inline:Nn \l_@@_rows_seq
3464 {
3465   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3466   \@@_line_with_light_syntax:n { #1 }
3467 }
3468 \tl_build_end:N \l_@@_new_body_tl
3469 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3470 {
3471   \int_set:Nn \l_@@_last_col_int
3472   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3473 }

```

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.

```

3474 \@@_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`).

```

3475 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3476 }
3477 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3478 {
3479   \seq_clear_new:N \l_@@_cells_seq
3480   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3481   \int_set:Nn \l_@@_nb_cols_int
3482   {
3483     \int_max:nn
3484     { \l_@@_nb_cols_int }
3485     { \seq_count:N \l_@@_cells_seq }
3486   }
3487   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3488   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3489   \seq_map_inline:Nn \l_@@_cells_seq
3490   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3491 }
3492 \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`.

```

3493 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3494 {
3495   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3496   { \@@_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.

```
3497   \end { #2 }
3498 }
```

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`).

```
3499 \cs_new:Npn \@@_create_col_nodes:
3500 {
3501   \crrc
3502   \int_if_zero:nT { \l_@@_first_col_int }
3503   {
3504     \omit
3505     \hbox_overlap_left:n
3506     {
3507       \bool_if:NT \l_@@_code_before_bool
3508       { \pgfsys@markposition { \@@_env: - col - 0 } }
3509       \pgfpicture
3510       \pgfrememberpicturepositiononpagetrue
3511       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3512       \str_if_empty:NF \l_@@_name_str
3513       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3514       \endpgfpicture
3515       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3516     }
3517     &
3518   }
3519   \omit
```

The following instruction must be put after the instruction `\omit` since, of course, it is not expandable.

```
3520   \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`).

```
3521   \int_if_zero:nTF { \l_@@_first_col_int }
3522   {
3523     \@@_mark_position:n { 1 }
3524     \pgfpicture
3525     \pgfrememberpicturepositiononpagetrue
3526     \pgfcoordinate { \@@_env: - col - 1 }
3527     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3528     \str_if_empty:NF \l_@@_name_str
3529     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3530     \endpgfpicture
3531   }
3532   {
3533     \bool_if:NT \l_@@_code_before_bool
3534     {
3535       \hbox
3536       {
3537         \skip_horizontal:n { 0.5 \arrayrulewidth }
3538         \pgfsys@markposition { \@@_env: - col - 1 }
3539         \skip_horizontal:n { -0.5 \arrayrulewidth }
3540       }
3541     }
3542     \pgfpicture
3543     \pgfrememberpicturepositiononpagetrue
3544     \pgfcoordinate { \@@_env: - col - 1 }
3545     { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3546     \@@_node_alias:n { 1 }
3547     \endpgfpicture
3548   }
```

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.

```

3549 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3550 \bool_if:NF \l_@@_auto_columns_width_bool
3551 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3552 {
3553   \bool_lazy_and:nnTF
3554     { \l_@@_auto_columns_width_bool }
3555     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3556     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3557     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3558   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3559 }
3560 \skip_horizontal:N \g_tmpa_skip
3561 \hbox
3562 {
3563   \@@_mark_position:n { 2 }
3564   \pgfpicture
3565   \pgfrememberpicturepositiononpagetrue
3566   \pgfcoordinate { \@@_env: - col - 2 }
3567     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3568   \@@_node_alias:n { 2 }
3569   \endpgfpicture
3570 }

```

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.

```

3571 \int_gset_eq:NN \g_tmpa_int \c_one_int
3572 \bool_if:NTF \g_@@_last_col_found_bool
3573 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3574 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3575 {
3576   &
3577   \omit
3578   \int_gincr:N \g_tmpa_int

```

The incrementation of the counter `\g_tmpa_int` must be done after the `\omit` of the cell.

```

3579 \skip_horizontal:N \g_tmpa_skip
3580 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

We create the col node on the right of the current column.

```

3581 \pgfpicture
3582 \pgfrememberpicturepositiononpagetrue
3583 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3584   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3585   \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3586 \endpgfpicture
3587 }

```

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).

```

3588 \bool_lazy_or:nnF
3589   { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3590   { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3591 {
3592   &
3593   \omit
3594   \skip_horizontal:N \g_tmpa_skip
3595   \int_gincr:N \g_tmpa_int
3596   \bool_lazy_any:nF

```

```

3597     {
3598         \g_@@_delims_bool
3599         \l_@@_tabular_bool
3600         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3601         \l_@@_exterior_arraycolsep_bool
3602         \l_@@_bar_at_end_of_pream_bool
3603     }
3604     { \skip_horizontal:n { - \col@sep } }
3605     \bool_if:NT \l_@@_code_before_bool
3606     {
3607         \hbox
3608         {
3609             \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.

```

3610         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3611         { \skip_horizontal:n { - \arraycolsep } }
3612         \pgfsys@markposition
3613         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3614         \skip_horizontal:n { 0.5 \arrayrulewidth }
3615         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3616         { \skip_horizontal:N \arraycolsep }
3617     }
3618 }
3619 \pgfpicture
3620 \pgfrememberpicturepositiononpagetrue
3621 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3622 {
3623     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3624     {
3625         \pgfpoint
3626         { - 0.5 \arrayrulewidth - \arraycolsep }
3627         \c_zero_dim
3628     }
3629     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3630 }
3631 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3632 \endpgfpicture
3633 }

3634 \bool_if:NT \g_@@_last_col_found_bool
3635 {
3636     \hbox_overlap_right:n
3637     {
3638         \skip_horizontal:N \g_@@_width_last_col_dim
3639         \skip_horizontal:N \col@sep
3640         \bool_if:NT \l_@@_code_before_bool
3641         {
3642             \pgfsys@markposition
3643             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3644         }
3645         \pgfpicture
3646         \pgfrememberpicturepositiononpagetrue
3647         \pgfcoordinate
3648         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3649         \pgfpointorigin
3650         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3651         \endpgfpicture
3652     }
3653 }
3654 }

```

```

3655 \cs_new_protected:Npn \@@_mark_position:n #1
3656 {
3657   \bool_if:NT \l_@@_code_before_bool
3658   {
3659     \hbox
3660     {
3661       \skip_horizontal:n { -0.5 \arrayrulewidth }
3662       \pgfsys@markposition { \@@_env: - col - #1 }
3663       \skip_horizontal:n { 0.5 \arrayrulewidth }
3664     }
3665   }
3666 }
3667 \cs_new_protected:Npn \@@_node_alias:n #1
3668 {
3669   \str_if_empty:NF \l_@@_name_str
3670   { \pgfnodelalias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3671 }

```

Here is the preamble for the “first column” (if the user uses the key `first-col`)

```

3672 \tl_const:Nn \c_@@_preamble_first_col_tl
3673 {
3674   >
3675   {

```

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

```

3676     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3677     \bool_gset_true:N \g_@@_after_col_zero_bool
3678     \@@_begin_of_row:
3679     \hbox_set:Nw \l_@@_cell_box
3680     \@@_math_toggle:
3681     \@@_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”.

```

3682     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3683     {
3684       \bool_lazy_or:nnT
3685       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3686       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3687       {
3688         \l_@@_code_for_first_col_tl
3689         \xglobal \colorlet { nicematrix-first-col } { . }
3690       }
3691     }
3692 }

```

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`.

```

3693   l
3694   <
3695   {
3696     \@@_math_toggle:
3697     \hbox_set_end:
3698     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3699     \@@_adjust_size_box:
3700     \@@_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.

```

3701     \dim_gset:Nn \g_@@_width_first_col_dim
3702     { \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.

```

3703     \hbox_overlap_left:n
3704     {
3705         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3706         { \@@_node_cell: }
3707         { \box_use_drop:N \l_@@_cell_box }
3708         \skip_horizontal:N \l_@@_left_delim_dim
3709         \skip_horizontal:N \l_@@_left_margin_dim
3710         \skip_horizontal:N \l_@@_extra_left_margin_dim
3711     }
3712     \bool_gset_false:N \g_@@_empty_cell_bool
3713     \skip_horizontal:n { -2 \col@sep }
3714 }
3715 }

```

Here is the preamble for the “last column” (if the user uses the key `last-col`).

```

3716 \tl_const:Nn \c_@@_preamble_last_col_tl
3717 {
3718     >
3719     {
3720         \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).

```

3721     \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.

```

3722     \bool_gset_true:N \g_@@_last_col_found_bool
3723     \int_gincr:N \c@jCol
3724     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3725     \hbox_set:Nw \l_@@_cell_box
3726     \@@_math_toggle:
3727     \@@_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”.

```

3728     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3729     {
3730         \bool_lazy_or:nnT
3731         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3732         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3733         {
3734             \l_@@_code_for_last_col_tl
3735             \xglobal \colorlet { nicematrix-last-col } { . }
3736         }
3737     }
3738 }
3739 1
3740 <
3741 {
3742     \@@_math_toggle:
3743     \hbox_set_end:
3744     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3745     \@@_adjust_size_box:
3746     \@@_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.

```

3747     \dim_gset:Nn \g_@@_width_last_col_dim
3748     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3749     \skip_horizontal:n { -2 \col@sep }

```

The content of the cell is inserted in an overlapping position.

```

3750     \hbox_overlap_right:n
3751     {

```

```

3752         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3753         {
3754             \skip_horizontal:N \l_@@_right_delim_dim
3755             \skip_horizontal:N \l_@@_right_margin_dim
3756             \skip_horizontal:N \l_@@_extra_right_margin_dim
3757             \@@_node_cell:
3758         }
3759     }
3760     \bool_gset_false:N \g_@@_empty_cell_bool
3761 }
3762 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}`.

```

3763 \NewDocumentEnvironment { NiceArray } { }
3764 {
3765     \bool_gset_false:N \g_@@_delims_bool
3766     \str_if_empty:NT \g_@@_name_env_str
3767     { \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).

```

3768     \NiceArrayWithDelims . .
3769 }
3770 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

3771 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3772 {
3773     \NewDocumentEnvironment { #1 NiceArray } { }
3774     {
3775         \bool_gset_true:N \g_@@_delims_bool
3776         \str_if_empty:NT \g_@@_name_env_str
3777         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3778         \@@_test_if_math_mode:
3779         \NiceArrayWithDelims #2 #3
3780     }
3781     { \endNiceArrayWithDelims }
3782 }
3783 \@@_def_env:NNN p ( )
3784 \@@_def_env:NNN b [ ]
3785 \@@_def_env:NNN B \{ \}
3786 \@@_def_env:NNN v \vert \vert
3787 \@@_def_env:NNN V \Vert \Vert

```

## 13 The environment `{NiceMatrix}` and its variants

```

3788 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3789 {
3790     \bool_set_false:N \l_@@_preamble_bool
3791     \tl_clear:N \l_tmpa_tl
3792     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3793     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3794     \tl_put_right:Nn \l_tmpa_tl
3795     {
3796         *
3797     {

```



```

3798 \int_case:nF \l_@@_last_col_int
3799 {
3800   { -2 } { \c@MaxMatrixCols }
3801   { -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).

```

3802   }
3803   { \int_eval:n { \l_@@_last_col_int - 1 } }
3804 }
3805 { #2 }
3806 }
3807 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3808 \exp_args:No \l_tmpb_tl \l_tpa_tl
3809 }
3810 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3811 \clist_map_inline:nn { p , b , B , v , V }
3812 {
3813   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3814   {
3815     \bool_gset_true:N \g_@@_delims_bool
3816     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3817     \int_if_zero:nT { \l_@@_last_col_int }
3818     {
3819       \bool_set_true:N \l_@@_last_col_without_value_bool
3820       \int_set:Nn \l_@@_last_col_int { -1 }
3821     }
3822     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3823     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3824   }
3825   { \use:c { end #1 NiceArray } }
3826 }

```

We define also an environment {NiceMatrix}

```

3827 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3828 {
3829   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3830   \int_if_zero:nT { \l_@@_last_col_int }
3831   {
3832     \bool_set_true:N \l_@@_last_col_without_value_bool
3833     \int_set:Nn \l_@@_last_col_int { -1 }
3834   }
3835   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3836   \bool_lazy_or:nnT
3837     { \clist_if_empty_p:N \l_@@_vlines_clist }
3838     { \l_@@_except_borders_bool }
3839     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3840   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3841 }
3842 { \endNiceArray }

```

The following command will be linked to \NotEmpty in the environments of nicematrix.

```

3843 \cs_new_protected:Npn \@@_NotEmpty:
3844 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## 14 {NiceTabular}, {NiceTabularX} and {NiceTabular\*}

```

3845 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3846 {

```

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.

```

3847 \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3848 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3849 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3850 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3851 \tl_if_empty:NF \l_@@_short_caption_tl
3852 {
3853   \tl_if_empty:NT \l_@@_caption_tl
3854   {
3855     \@@_error_or_warning:n { short-caption-without~caption }
3856     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3857   }
3858 }
3859 \tl_if_empty:NF \l_@@_label_tl
3860 {
3861   \tl_if_empty:NT \l_@@_caption_tl
3862   { \@@_error_or_warning:n { label~without~caption } }
3863 }
3864 \NewDocumentEnvironment { TabularNote } { b }
3865 {
3866   \bool_if:NTF \l_@@_in_code_after_bool
3867   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3868   {
3869     \tl_if_empty:NF \g_@@_tabularnote_tl
3870     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3871     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3872   }
3873 }
3874 { }
3875 \@@_settings_for_tabular:
3876 \NiceArray { #2 }
3877 }
3878 { \endNiceArray }
3879 \cs_new_protected:Npn \@@_settings_for_tabular:
3880 {
3881   \bool_set_true:N \l_@@_tabular_bool
3882   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3883   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3884   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3885 }

3886 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3887 {
3888   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3889   \dim_set:Nn \l_@@_width_dim { #1 }
3890   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3891   \@@_settings_for_tabular:
3892   \NiceArray { #3 }
3893 }
3894 {
3895   \endNiceArray
3896   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3897   { \@@_error:n { NiceTabularX~without~X } }
3898 }

3899 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3900 {
3901   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3902   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3903   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3904   \@@_settings_for_tabular:
3905   \NiceArray { #3 }
3906 }
3907 { \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`).

```

3908 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3909 {
3910   \bool_lazy_all:nT
3911   {
3912     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3913     { \l_@@_hvlines_bool }
3914     { ! \g_@@_delims_bool }
3915     { ! \l_@@_except_borders_bool }
3916   }
3917   {
3918     \bool_set_true:N \l_@@_except_borders_bool
3919     \clist_if_empty:NF \l_@@_corners_clist
3920     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3921     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3922     {
3923       \@@_stroke_block:nnn
3924       {
3925         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3926         draw = \l_@@_rules_color_tl
3927       }
3928       { 1-1 }
3929       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3930     }
3931   }
3932 }

3933 \cs_new_protected:Npn \@@_after_array:
3934 {

```

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`.

```

3935   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3936   \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.

```

3937   \bool_if:NT \g_@@_last_col_found_bool
3938   { \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`.

```

3939   \bool_if:NT \l_@@_last_col_without_value_bool
3940   { \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.

```

3941   \bool_if:NT \l_@@_last_row_without_value_bool
3942   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3943 \tl_gput_right:Ne \g_@@_aux_tl
3944 {
3945   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3946   {
3947     \int_use:N \l_@@_first_row_int ,
3948     \int_use:N \c@iRow ,
3949     \int_use:N \g_@@_row_total_int ,
3950     \int_use:N \l_@@_first_col_int ,
3951     \int_use:N \c@jCol ,
3952     \int_use:N \g_@@_col_total_int
3953   }
3954 }

```

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`).

```

3955 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3956 {
3957   \tl_gput_right:Ne \g_@@_aux_tl
3958   {
3959     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3960     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3961   }
3962 }
3963 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3964 {
3965   \tl_gput_right:Ne \g_@@_aux_tl
3966   {
3967     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3968     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3969     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3970     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3971   }
3972 }

```

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

```

3973 \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3974 \pgfpicture
3975 \@@_create_aliases_last:
3976 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3977 \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.

```

3978 \bool_if:NT \l_@@_parallelize_diags_bool
3979 {
3980   \int_gzero:N \g_@@_ddots_int
3981   \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.

```

3982   \dim_gzero:N \g_@@_delta_x_one_dim
3983   \dim_gzero:N \g_@@_delta_y_one_dim
3984   \dim_gzero:N \g_@@_delta_x_two_dim
3985   \dim_gzero:N \g_@@_delta_y_two_dim
3986 }

```

---

<sup>12</sup>It's possible to use the option `parallelize-diags` to disable this parallelization.

```

3987 \bool_set_false:N \l_@@_initial_open_bool
3988 \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.

```

3989 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

3990 \@@_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_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3991 \clist_if_empty:NF \l_@@_corners_clist
3992 {
3993   \bool_if:NTF \l_@@_no_cell_nodes_bool
3994     { \@@_error:n { corners~with~no~cell~nodes } }
3995     { \@@_compute_corners: }
3996 }

```

By design, we have computed the corners before the adjunction of `\g_@@_future_pos_of_blocks_seq` is used by `\EmptyRow` and `\EmptyColumn` in the `\CodeBefore`.

```

3997 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
3998   \g_@@_pos_of_blocks_seq
3999   \g_@@_future_pos_of_blocks_seq
4000 \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-*}`).

```

4001 \@@_adjust_pos_of_blocks_seq:
4002 \@@_deal_with_rounded_corners:
4003 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
4004 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

Now, the pre-code-after and then, the `\CodeAfter`.

```

4005 \IfPackageLoadedT { tikz }
4006 {
4007   \tikzset
4008   {
4009     every~picture / .style =
4010     {
4011       overlay ,
4012       remember~picture ,
4013       name~prefix = \@@_env: -
4014     }
4015   }
4016 }
4017 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4018 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4019 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4020 \cs_set_eq:NN \OverBrace \@@_OverBrace
4021 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4022 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4023 \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.

```

4024 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
4025 \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.

```
4026 \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).

```
4027 \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`).

```
4028 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4029 { \@@_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:`.

```
4030 \bool_set_true:N \l_@@_in_code_after_bool
4031 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4032 \scan_stop:
4033 \tl_gclear:N \g_nicematrix_code_after_tl
4034 \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.

```
4035 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4036 \tl_if_empty:NF \g_@@_pre_code_before_tl
4037 {
4038   \tl_gput_right:Ne \g_@@_aux_tl
4039   {
4040     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4041     { \exp_not:o \g_@@_pre_code_before_tl }
4042   }
4043   \tl_gclear:N \g_@@_pre_code_before_tl
4044 }
4045 \tl_if_empty:NF \g_nicematrix_code_before_tl
4046 {
4047   \tl_gput_right:Ne \g_@@_aux_tl
4048   {
4049     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4050     { \exp_not:o \g_nicematrix_code_before_tl }
4051   }
4052   \tl_gclear:N \g_nicematrix_code_before_tl
4053 }

4054 \str_gclear:N \g_@@_name_env_str
4055 \@@_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`.

```
4056 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4057 }
```

---

<sup>13</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

4058 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4059 {
4060   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4061   \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.

```

4062   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4063     { 0.6 \l_@@_xdots_shorten_start_dim }
4064   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4065     { 0.6 \l_@@_xdots_shorten_end_dim }
4066 }

```

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`.

```

4067 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4068 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4069 \cs_new_protected:Npn \@@_create_alias_nodes:
4070 {
4071   \int_step_inline:nn { \c@iRow }
4072   {
4073     \pgfnodealias
4074       { \l_@@_name_str - ##1 - last }
4075       { \@@_env: - ##1 - \int_use:N \c@jCol }
4076   }
4077   \int_step_inline:nn { \c@jCol }
4078   {
4079     \pgfnodealias
4080       { \l_@@_name_str - last - ##1 }
4081       { \@@_env: - \int_use:N \c@iRow - ##1 }
4082   }
4083   \pgfnodealias
4084     { \l_@@_name_str - last - last }
4085     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4086 }

```

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).

```

4087 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4088 {
4089   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4090     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4091 }

```

The following command must *not* be protected.

```

4092 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4093 {
4094   { #1 }
4095   { #2 }
4096   {
4097     \int_compare:nNnTF { #3 } > { 98 }
4098       { \int_use:N \c@iRow }
4099       { #3 }
4100   }

```

```

4101 {
4102   \int_compare:nNnTF { #4 } > { 98 }
4103     { \int_use:N \c@jCol }
4104     { #4 }
4105 }
4106 { #5 }
4107 }

```

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).

```

4108 \hook_gput_code:nnn { begindocument } { . }
4109 {
4110   \cs_new_protected:Npe \@@_draw_dotted_lines:
4111     {
4112       \c_@@_pgfortikzpicture_tl
4113       \@@_draw_dotted_lines_i:
4114       \c_@@_endpgfortikzpicture_tl
4115     }
4116 }

```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```

4117 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4118 {
4119   \pgfrememberpicturepositiononpagetrue
4120   \pgf@relevantforpicturesizefalse
4121   \g_@@_HVdotsfor_lines_tl
4122   \g_@@_Vdots_lines_tl
4123   \g_@@_Ddots_lines_tl
4124   \g_@@_Iddots_lines_tl
4125   \g_@@_Cdots_lines_tl
4126   \g_@@_Ldots_lines_tl
4127 }

4128 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4129 {
4130   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4131   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4132 }

```

We define a new PGF shape for the diag nodes because we want to provide an anchor called `.5` for those nodes.

```

4133 \pgfdeclareshape { @@_diag_node }
4134 {
4135   \savedanchor { \five }
4136   {
4137     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4138     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4139   }
4140   \anchor { 5 } { \five }
4141   \anchor { center } { \pgfpointorigin }
4142   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4143   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4144   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4145   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4146   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4147   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4148   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4149   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4150   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4151   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4152 }

```



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).

```

4153 \cs_new_protected:Npn \@@_create_diag_nodes:
4154 {
4155   \pgfpicture
4156   \pgfrememberpicturepositiononpagetrue
4157   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4158   {
4159     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4160     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4161     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4162     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4163     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4164     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4165     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4166     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4167     \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.

```

4168 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4169 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4170 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4171 \str_if_empty:NF \l_@@_name_str
4172 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4173 }

```

Now, the last node. Of course, that is only a `coordinate` because there is not `.5` anchor for that node.

```

4174 \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4175 @@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4176 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4177 @@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4178 \pgfcoordinate
4179 { @@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4180 \pgfnodealias
4181 { @@_env: - last }
4182 { @@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4183 \str_if_empty:NF \l_@@_name_str
4184 {
4185   \pgfnodealias
4186   { \l_@@_name_str - \int_use:N \l_tmpa_int }
4187   { @@_env: - \int_use:N \l_tmpa_int }
4188   \pgfnodealias
4189   { \l_@@_name_str - last }
4190   { @@_env: - last }
4191 }
4192 \endpgfpicture
4193 }

```

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.

$$\begin{pmatrix} a+b+c & a+b & a \\ a & \dots & \dots \\ a & a+b & a+b+c \end{pmatrix}$$

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.

```

4194 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4195 {

```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```

4196   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4197   \l_@@_initial_i_int = #1
4198   \l_@@_initial_j_int = #2
4199   \l_@@_final_i_int = #1
4200   \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.

```

4201     \let \l_@@_stop_loop_bool \c_false_bool
4202     \bool_do_until:Nn \l_@@_stop_loop_bool
4203     {

```

We test if we are still in the matrix.

```

4204         \advance \l_@@_final_i_int by #3
4205         \advance \l_@@_final_j_int by #4
4206         \let \l_@@_final_open_bool \c_false_bool
4207         \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4208             \if_int_compare:w #3 = \c_one_int
4209                 \let \l_@@_final_open_bool \c_true_bool
4210         \else:
4211             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4212                 \let \l_@@_final_open_bool \c_true_bool
4213         \fi:
4214     \fi:
4215 \else:
4216     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4217         \if_int_compare:w #4 = -1
4218             \let \l_@@_final_open_bool \c_true_bool
4219         \fi:
4220     \else:
4221         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4222             \if_int_compare:w #4 = \c_one_int
4223                 \let \l_@@_final_open_bool \c_true_bool
4224             \fi:
4225         \fi:
4226     \fi:
4227 \fi:
4228 \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.

```

4229     {

```

We do a step backwards.

```

4230         \advance \l_@@_final_i_int by - #3
4231         \advance \l_@@_final_j_int by - #4
4232         \let \l_@@_stop_loop_bool \c_true_bool
4233     }

```

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`.

```

4234     {
4235         \cs_if_exist:cTF
4236         {
4237             @@ _ dotted _
4238             \int_use:N \l_@@_final_i_int -
4239             \int_use:N \l_@@_final_j_int
4240         }
4241         {
4242             \advance \l_@@_final_i_int by - #3
4243             \advance \l_@@_final_j_int by - #4
4244             \let \l_@@_final_open_bool \c_true_bool
4245             \let \l_@@_stop_loop_bool \c_true_bool
4246         }
4247     {
4248         \cs_if_exist:cTF
4249         {
4250             pgf @ sh @ ns @ \@@_env:
4251             - \int_use:N \l_@@_final_i_int
4252             - \int_use:N \l_@@_final_j_int
4253         }

```

```

4254         { \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.

```

4255     {
4256         \cs_set_nopar:cpn
4257         {
4258             @@ _ dotted _
4259             \int_use:N \l_@@_final_i_int -
4260             \int_use:N \l_@@_final_j_int
4261         }
4262     { }
4263 }
4264 }
4265 }
4266 }

```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```

4267     \let \l_@@_stop_loop_bool \c_false_bool

```

The following line of code is only for efficiency in the following loop.

```

4268     \l_tmpa_int = \l_@@_col_min_int
4269     \advance \l_tmpa_int by -1
4270     \bool_do_until:Nn \l_@@_stop_loop_bool
4271     {
4272         \advance \l_@@_initial_i_int by - #3
4273         \advance \l_@@_initial_j_int by - #4
4274         \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.

```

4275         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4276         \if_int_compare:w #3 = \c_one_int
4277             \let \l_@@_initial_open_bool \c_true_bool
4278         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4279         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4280             \let \l_@@_initial_open_bool \c_true_bool
4281         \fi:
4282     \fi:
4283 \else:
4284     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4285     \if_int_compare:w #4 = \c_one_int
4286         \let \l_@@_initial_open_bool \c_true_bool
4287     \fi:
4288 \else:
4289     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4290     \if_int_compare:w #4 = -1
4291         \let \l_@@_initial_open_bool \c_true_bool
4292     \fi:
4293     \fi:
4294 \fi:
4295 \fi:
4296 \bool_if:NTF \l_@@_initial_open_bool
4297 {
4298     \advance \l_@@_initial_i_int by #3
4299     \advance \l_@@_initial_j_int by #4

```

```

4300     \let \l_@@_stop_loop_bool \c_true_bool
4301   }
4302   {
4303     \cs_if_exist:cTF
4304     {
4305       @@ _ dotted _
4306       \int_use:N \l_@@_initial_i_int -
4307       \int_use:N \l_@@_initial_j_int
4308     }
4309     {
4310       \advance \l_@@_initial_i_int by #3
4311       \advance \l_@@_initial_j_int by #4
4312       \let \l_@@_initial_open_bool \c_true_bool
4313       \let \l_@@_stop_loop_bool \c_true_bool
4314     }
4315     {
4316       \cs_if_exist:cTF
4317       {
4318         pgf @ sh @ ns @ \@@_env:
4319         - \int_use:N \l_@@_initial_i_int
4320         - \int_use:N \l_@@_initial_j_int
4321       }
4322       { \let \l_@@_stop_loop_bool \c_true_bool }
4323       {
4324         \cs_set_nopar:cpn
4325         {
4326           @@ _ dotted _
4327           \int_use:N \l_@@_initial_i_int -
4328           \int_use:N \l_@@_initial_j_int
4329         }
4330         { }
4331       }
4332     }
4333   }
4334 }

```

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.

```

4335   \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4336   {
4337     { \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`.

```

4338     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4339     { \int_use:N \l_@@_final_i_int }
4340     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4341     { }
4342   }
4343 }

```

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.

```

4344   \cs_new_protected:Npn \@@_open_shorten:
4345   {
4346     \bool_if:NT \l_@@_initial_open_bool
4347     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4348     \bool_if:NT \l_@@_final_open_bool
4349     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4350   }

```

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).

```

4351 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4352 {
4353   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4354   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4355   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4356   \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`.

```

4357   \seq_if_empty:NF \g_@@_submatrix_seq
4358   {
4359     \seq_map_inline:Nn \g_@@_submatrix_seq
4360     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4361   }
4362 }

```

#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.

```

4363 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4364 {
4365   \if_int_compare:w #3 > #1
4366   \else:
4367     \if_int_compare:w #1 > #5
4368     \else:
4369       \if_int_compare:w #4 > #2
4370       \else:
4371         \if_int_compare:w #2 > #6
4372         \else:
4373           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4374           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4375           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4376           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4377         \fi:
4378       \fi:
4379     \fi:
4380   \fi:
4381 }

```



```

4382 \cs_new_protected:Npn \@@_set_initial_coords:
4383 {
4384   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4385   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4386 }
4387 \cs_new_protected:Npn \@@_set_final_coords:
4388 {
4389   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4390   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4391 }
4392 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4393 {
4394   \pgfpointanchor
4395   {
4396     \@@_env:
4397     - \int_use:N \l_@@_initial_i_int
4398     - \int_use:N \l_@@_initial_j_int
4399   }
4400   { #1 }
4401   \@@_set_initial_coords:
4402 }
4403 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4404 {
4405   \pgfpointanchor
4406   {
4407     \@@_env:
4408     - \int_use:N \l_@@_final_i_int
4409     - \int_use:N \l_@@_final_j_int
4410   }
4411   { #1 }
4412   \@@_set_final_coords:
4413 }
4414 \cs_new_protected:Npn \@@_open_x_initial_dim:
4415 {
4416   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4417   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4418   {
4419     \cs_if_exist:cT
4420     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4421     {
4422       \pgfpointanchor
4423       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4424       { west }
4425       \dim_set:Nn \l_@@_x_initial_dim
4426       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4427     }
4428   }

```

If, in fact, all the cells of the column are empty (no PGF/TikZ nodes in those cells).

```

4429   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4430   {
4431     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4432     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4433     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4434   }
4435 }
4436 \cs_new_protected:Npn \@@_open_x_final_dim:
4437 {
4438   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4439   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4440   {
4441     \cs_if_exist:cT
4442     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4443     {
4444         \pgfpointanchor
4445         { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4446         { east }
4447         \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4448         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4449     }
4450 }

```

If, in fact, all the cells of the columns are empty (no PGF/TikZ nodes in those cells).

```

4451     \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4452     {
4453         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4454         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4455         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4456     }
4457 }

```

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.

```

4458 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4459 {
4460     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4461     \cs_if_free:cT { @ _ dotted _ #1 - #2 }
4462     {
4463         \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.

```

4464     \bool_if:NT \g_@@_aux_found_bool
4465     {
4466         \group_begin:
4467         \l_@@_open_shorten:
4468         \int_if_zero:nTF { #1 }
4469         { \color { nicematrix-first-row } }
4470     }

```

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.

```

4471         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4472         { \color { nicematrix-last-row } }
4473     }
4474     \keys_set:nn { nicematrix / xdots } { #3 }
4475     \l_@@_color:o \l_@@_xdots_color_tl
4476     \l_@@_actually_draw_Ldots:
4477     \group_end:
4478 }
4479 }
4480 }

```

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`.

```

4481 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4482 {
4483   \bool_if:NTF \l_@@_initial_open_bool
4484   {
4485     \@@_open_x_initial_dim:
4486     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4487     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4488   }
4489   { \@@_set_initial_coords_from_anchor:n { base-east } }
4490   \bool_if:NTF \l_@@_final_open_bool
4491   {
4492     \@@_open_x_final_dim:
4493     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4494     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4495   }
4496   { \@@_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.

```

4497   \bool_lazy_all:nTF
4498   {
4499     \l_@@_initial_open_bool
4500     \l_@@_final_open_bool
4501     { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4502   }
4503   {
4504     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4505     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4506   }

```

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 (?).

```

4507   {
4508     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4509     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4510   }
4511   \@@_draw_line:
4512 }

```

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.

```

4513 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4514 {
4515   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4516   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4517   {
4518     \@@_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.

```

4519   \bool_if:NT \g_@@_aux_found_bool
4520   {
4521     \group_begin:
4522     \@@_open_shorten:
4523     \int_if_zero:nTF { #1 }
4524     { \color { nicematrix-first-row } }
4525     {

```

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.

```

4526     \int_compare:nNnT { #1 } = { \l_@@_last_row_int }

```

```

4527         { \color { nicematrix-last-row } }
4528     }
4529     \keys_set:nn { nicematrix / xdots } { #3 }
4530     \@@_color:o \l_@@_xdots_color_tl
4531     \@@_actually_draw_Cdots:
4532 \group_end:
4533 }
4534 }
4535 }

```

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.`

```

4536 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4537 {
4538     \bool_if:NTF \l_@@_initial_open_bool
4539     { \@@_open_x_initial_dim: }
4540     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4541     \bool_if:NTF \l_@@_final_open_bool
4542     { \@@_open_x_final_dim: }
4543     { \@@_set_final_coords_from_anchor:n { mid-west } }
4544     \bool_lazy_and:nnTF
4545     { \l_@@_initial_open_bool }
4546     { \l_@@_final_open_bool }
4547     {
4548         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4549         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4550         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4551         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4552         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4553     }
4554     {
4555         \bool_if:NT \l_@@_initial_open_bool
4556         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4557         \bool_if:NT \l_@@_final_open_bool
4558         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4559     }
4560     \@@_draw_line:
4561 }
4562 \cs_new_protected:Npn \@@_open_y_initial_dim:
4563 {
4564     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4565     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4566     {
4567         \cs_if_exist:cT
4568         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4569         {
4570             \pgfpointanchor
4571             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4572             { north }
4573             \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4574             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4575         }
4576     }

```

```

4577 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4578 {
4579   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4580   \dim_set:Nn \l_@@_y_initial_dim
4581   {
4582     \fp_to_dim:n
4583     {
4584       \pgf@y
4585       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4586     }
4587   }
4588 }
4589 }
4590 \cs_new_protected:Npn \@@_open_y_final_dim:
4591 {
4592   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4593   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4594   {
4595     \cs_if_exist:cT
4596     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4597     {
4598       \pgfpointanchor
4599       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4600       { south }
4601       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4602       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4603     }
4604   }
4605   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4606   {
4607     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4608     \dim_set:Nn \l_@@_y_final_dim
4609     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4610   }
4611 }

```

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.

```

4612 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4613 {
4614   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4615   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4616   {
4617     \@@_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.

```

4618   \bool_if:NT \g_@@_aux_found_bool
4619   {
4620     \group_begin:
4621     \@@_open_shorten:
4622     \int_if_zero:nTF { #2 }
4623     { \color { nicematrix-first-col } }
4624     {
4625       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4626       { \color { nicematrix-last-col } }
4627     }
4628     \keys_set:nn { nicematrix / xdots } { #3 }
4629     \@@_color:o \l_@@_xdots_color_tl
4630     \bool_if:NTF \l_@@_Vbrace_bool
4631     { \@@_actually_draw_Vbrace: }
4632     { \@@_actually_draw_Vdots: }
4633     \group_end:
4634   }

```

```

4635     }
4636 }

```

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`.

```

4637 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4638 {
4639   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4640   { \@@_actually_draw_Vdots_i: }
4641   { \@@_actually_draw_Vdots_ii: }
4642   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4643   \@@_draw_line:
4644 }

```

First, the case of a dotted line open on both sides.

```

4645 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4646 {
4647   \@@_open_y_initial_dim:
4648   \@@_open_y_final_dim:
4649   \int_if_zero:nTF { \l_@@_initial_j_int }

```

We have a dotted line open on both sides in the “first column”.

```

4650   {
4651     \@@_qpoint:n { col - 1 }
4652     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4653     \dim_sub:Nn \l_@@_x_initial_dim
4654     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4655   }
4656   {
4657     \bool_lazy_and:nnTF
4658     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4659     {
4660       \int_compare_p:nNn
4661       { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4662     }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4663   {
4664     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4665     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4666     \dim_add:Nn \l_@@_x_initial_dim
4667     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4668   }

```

We have a dotted line open on both sides which is *not* in an exterior column.

```

4669   {
4670     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4671     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4672     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4673     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4674   }
4675 }
4676 }

```

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 1 or may be considered as if.

```

4677 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4678 {
4679   \bool_set_false:N \l_tmpa_bool
4680   \bool_if:NF \l_@@_initial_open_bool
4681   {
4682     \bool_if:NF \l_@@_final_open_bool
4683     {
4684       \@@_set_initial_coords_from_anchor:n { south~west }
4685       \@@_set_final_coords_from_anchor:n { north~west }
4686       \bool_set:Nn \l_tmpa_bool
4687       {
4688         \dim_compare_p:nNn
4689         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4690       }
4691     }
4692   }

```

Now, we try to determine whether the column is of type c or may be considered as if.

```

4693   \bool_if:NTF \l_@@_initial_open_bool
4694   {
4695     \@@_open_y_initial_dim:
4696     \@@_set_final_coords_from_anchor:n { north }
4697     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4698   }
4699   {
4700     \@@_set_initial_coords_from_anchor:n { south }
4701     \bool_if:NTF \l_@@_final_open_bool
4702     { \@@_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.

```

4703   {
4704     \@@_set_final_coords_from_anchor:n { north }
4705     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4706     {
4707       \dim_set:Nn \l_@@_x_initial_dim
4708       {
4709         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4710         \l_@@_x_initial_dim \l_@@_x_final_dim
4711       }
4712     }
4713   }
4714 }
4715 }

```

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`.

```

4716 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4717 {
4718   \bool_if:NTF \l_@@_initial_open_bool
4719     { \@@_open_y_initial_dim: }
4720     { \@@_set_initial_coords_from_anchor:n { south } }
4721   \bool_if:NTF \l_@@_final_open_bool
4722     { \@@_open_y_final_dim: }
4723     { \@@_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.

```

4724   \int_if_zero:nTF { \l_@@_initial_j_int }
4725   {
4726     \@@_qpoint:n { col - 1 }
4727     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4728     \dim_sub:Nn \l_@@_x_initial_dim
4729       { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4730   }

```

Elsewhere, the brace must be drawn left flush.

```

4731   {
4732     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4733     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4734     \dim_add:Nn \l_@@_x_initial_dim
4735       { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4736   }

```

We draw a vertical rule and that's why, of course, both  $x$ -values are equal.

```

4737   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4738   \@@_draw_line:
4739 }

```

```

4740 \cs_new:Npn \@@_colsep:
4741 { \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.

```

4742 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4743 {
4744   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4745   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4746   {
4747     \@@_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.

```

4748   \bool_if:NT \g_@@_aux_found_bool
4749   {
4750     \group_begin:
4751       \@@_open_shorten:
4752       \keys_set:nn { nicematrix / xdots } { #3 }
4753       \@@_color:o \l_@@_xdots_color_tl
4754       \@@_actually_draw_Ddots:
4755     \group_end:
4756   }
4757 }
4758 }

```



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.`

```

4759 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4760 {
4761   \bool_if:NTF \l_@@_initial_open_bool
4762   {
4763     \@@_open_y_initial_dim:
4764     \@@_open_x_initial_dim:
4765   }
4766   { \@@_set_initial_coords_from_anchor:n { south~east } }
4767   \bool_if:NTF \l_@@_final_open_bool
4768   {
4769     \@@_open_x_final_dim:
4770     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4771   }
4772   { \@@_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.

```

4773   \bool_if:NT \l_@@_parallelize_diags_bool
4774   {
4775     \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).

```

4776     \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.

```

4777   {
4778     \dim_gset:Nn \g_@@_delta_x_one_dim
4779     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4780     \dim_gset:Nn \g_@@_delta_y_one_dim
4781     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4782   }

```

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`.

```

4783   {
4784     \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4785     {
4786       \dim_set:Nn \l_@@_y_final_dim
4787       {
4788         \l_@@_y_initial_dim +
4789         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4790         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4791       }
4792     }
4793   }
4794 }
4795 \@@_draw_line:
4796 }

```

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.

```

4797 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4798 {
4799   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4800   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4801   {
4802     \@@_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.

```

4803     \bool_if:NT \g_@@_aux_found_bool
4804     {
4805       \group_begin:
4806       \@@_open_shorten:
4807       \keys_set:nn { nicematrix / xdots } { #3 }
4808       \@@_color:o \l_@@_xdots_color_tl
4809       \@@_actually_draw_Iddots:
4810       \group_end:
4811     }
4812   }
4813 }

```

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`.

```

4814 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4815 {
4816   \bool_if:NTF \l_@@_initial_open_bool
4817   {
4818     \@@_open_y_initial_dim:
4819     \@@_open_x_initial_dim:
4820   }
4821   { \@@_set_initial_coords_from_anchor:n { south-west } }
4822   \bool_if:NTF \l_@@_final_open_bool
4823   {
4824     \@@_open_y_final_dim:
4825     \@@_open_x_final_dim:
4826   }
4827   { \@@_set_final_coords_from_anchor:n { north-east } }
4828   \bool_if:NT \l_@@_parallelize_diags_bool
4829   {
4830     \int_gincr:N \g_@@_iddots_int
4831     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4832     {
4833       \dim_gset:Nn \g_@@_delta_x_two_dim
4834       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4835       \dim_gset:Nn \g_@@_delta_y_two_dim
4836       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4837     }
4838     {
4839       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4840       {

```

```

4841         \dim_set:Nn \l_@@_y_final_dim
4842         {
4843             \l_@@_y_initial_dim +
4844             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4845             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4846         }
4847     }
4848 }
4849 }
4850 \@@_draw_line:
4851 }

```

## 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`

```

4852 \cs_new_protected:Npn \@@_draw_line:
4853 {
4854     \pgfrememberpicturepositiononpagetrue
4855     \pgf@relevantforpicturesizefalse
4856     \bool_lazy_or:nnTF
4857     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4858     { \l_@@_dotted_bool }
4859     { \@@_draw_standard_dotted_line: }
4860     { \@@_draw_unstandard_dotted_line: }
4861 }

```

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.

```

4862 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4863 {
4864     \begin { scope }
4865     \@@_draw_unstandard_dotted_line:o
4866     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4867 }

```

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.

```

4868 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4869 {
4870     \@@_draw_unstandard_dotted_line:nooo
4871     { #1 }
4872     \l_@@_xdots_up_tl
4873     \l_@@_xdots_down_tl
4874     \l_@@_xdots_middle_tl
4875 }
4876 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following TikZ styles are for the three labels (set by the symbols  $_$ ,  $^$  and  $=$ ) of a continuous line with a non-standard style.

```

4877 \hook_gput_code:nnn { begindocument } { . }
4878 {
4879   \IfPackageLoadedT { tikz }
4880   {
4881     \tikzset
4882     {
4883       @@_node_above / .style = { sloped , above } ,
4884       @@_node_below / .style = { sloped , below } ,
4885       @@_node_middle / .style =
4886       {
4887         sloped ,
4888         inner-sep = \c_@@_innersep_middle_dim
4889       }
4890     }
4891   }
4892 }

4893 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4894 {

```

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.

```

4895   \dim_zero_new:N \l_@@_l_dim
4896   \dim_set:Nn \l_@@_l_dim
4897   {
4898     \fp_to_dim:n
4899     {
4900       sqrt
4901       (
4902         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4903         +
4904         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4905       )
4906     }
4907   }

```

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.

```

4908   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4909   {
4910     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4911     \@@_draw_unstandard_dotted_line_i:
4912   }

```

If the key `xdots/horizontal-labels` has been used.

```

4913   \bool_if:NT \l_@@_xdots_h_labels_bool
4914   {
4915     \tikzset
4916     {
4917       @@_node_above / .style = { auto = left } ,
4918       @@_node_below / .style = { auto = right } ,
4919       @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4920     }
4921   }
4922   \tl_if_empty:nF { #4 }
4923   { \tikzset { @@_node_middle / .append~style = { fill = white } } }

```

```

4924 \dim_zero:N \l_tmpa_dim
4925 \dim_zero:N \l_tmpb_dim
4926 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4927 {

```

We test whether the brace is vertical or horizontal.

```

4928 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4929 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4930 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4931 }
4932 {
4933 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4934 {
4935 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4936 { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4937 { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4938 }
4939 }
4940 \use:e
4941 {
4942 \exp_not:N \begin { scope }
4943 [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
4944 }
4945 \draw
4946 [ #1 ]
4947 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4948 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4949 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4950 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4951 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4952 \end { scope }
4953 \end { scope }
4954 }
4955 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4956 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4957 {
4958 \dim_set:Nn \l_tmpa_dim
4959 {
4960 \l_@@_x_initial_dim
4961 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4962 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4963 }
4964 \dim_set:Nn \l_tmpb_dim
4965 {
4966 \l_@@_y_initial_dim
4967 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4968 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4969 }
4970 \dim_set:Nn \l_@@_tmpc_dim
4971 {
4972 \l_@@_x_final_dim
4973 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4974 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4975 }
4976 \dim_set:Nn \l_@@_tmpd_dim
4977 {
4978 \l_@@_y_final_dim
4979 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4980 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4981 }
4982 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4983 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4984 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim

```

```

4985 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4986 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4987 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4988 {
4989   \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.

```

4990   \dim_zero_new:N \l_@@_l_dim
4991   \dim_set:Nn \l_@@_l_dim
4992   {
4993     \fp_to_dim:n
4994     {
4995       sqrt
4996       (
4997         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4998         +
4999         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5000       )
5001     }
5002   }

```

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.

```

5003   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
5004   {
5005     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
5006     { \@@_draw_standard_dotted_line_i: }
5007   }
5008   \group_end:
5009   \bool_lazy_all:nF
5010   {
5011     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5012     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5013     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5014   }
5015   { \@@_labels_standard_dotted_line: }
5016 }
5017 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5018 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5019 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

5020   \int_set:Nn \l_tmpa_int
5021   {
5022     \dim_ratio:nn
5023     {
5024       \l_@@_l_dim
5025       - \l_@@_xdots_shorten_start_dim
5026       - \l_@@_xdots_shorten_end_dim
5027     }
5028     { \l_@@_xdots_inter_dim }
5029   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

5030   \dim_set:Nn \l_tmpa_dim

```

```

5031 {
5032   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5033   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5034 }
5035 \dim_set:Nn \l_tmpb_dim
5036 {
5037   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5038   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5039 }

```

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.

```

5040 \dim_gadd:Nn \l_@@_x_initial_dim
5041 {
5042   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5043   \dim_ratio:nn
5044   {
5045     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5046     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5047   }
5048   { 2 \l_@@_l_dim }
5049 }
5050 \dim_gadd:Nn \l_@@_y_initial_dim
5051 {
5052   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5053   \dim_ratio:nn
5054   {
5055     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5056     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5057   }
5058   { 2 \l_@@_l_dim }
5059 }
5060 \pgf@relevantforpicturesizefalse
5061 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
5062 {
5063   \pgfpathcircle
5064   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5065   { \l_@@_xdots_radius_dim }
5066   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5067   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5068 }
5069 \pgfusepathqfill
5070 }

```

```

5071 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5072 {
5073   \pgfscope
5074   \pgftransformshift
5075   {
5076     \pgfpointlineatime { 0.5 }
5077     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5078     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5079   }
5080   \fp_set:Nn \l_tmpa_fp
5081   {
5082     atand
5083     (
5084       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5085       \l_@@_x_final_dim - \l_@@_x_initial_dim
5086     )
5087   }
5088   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5089   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }

```

```

5090 \tl_if_empty:NF \l_@@_xdots_middle_tl
5091 {
5092   \begin { pgfscope }
5093   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5094   \pgfnode
5095     { rectangle }
5096     { center }
5097     {
5098       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5099       {
5100         $ % $
5101         \scriptstyle \l_@@_xdots_middle_tl
5102         $ % $
5103       }
5104     }
5105     { }
5106     {
5107       \pgfsetfillcolor { white }
5108       \pgfusepath { fill }
5109     }
5110   \end { pgfscope }
5111 }
5112 \tl_if_empty:NF \l_@@_xdots_up_tl
5113 {
5114   \pgfnode
5115     { rectangle }
5116     { south }
5117     {
5118       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5119       {
5120         $ % $
5121         \scriptstyle \l_@@_xdots_up_tl
5122         $ % $
5123       }
5124     }
5125     { }
5126     { \pgfusepath { } }
5127 }
5128 \tl_if_empty:NF \l_@@_xdots_down_tl
5129 {
5130   \pgfnode
5131     { rectangle }
5132     { north }
5133     {
5134       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5135       {
5136         $ % $
5137         \scriptstyle \l_@@_xdots_down_tl
5138         $ % $
5139       }
5140     }
5141     { }
5142     { \pgfusepath { } }
5143 }
5144 \endpgfscope
5145 }

```



## 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.

```
5146 \hook_gput_code:nnn { begindocument } { . }
5147 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
5148 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5149 \cs_new_protected:Npn \@@_Ldots:
5150 { \@@_collect_options:n { \@@_Ldots_i } }
5151 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5152 {
5153   \int_if_zero:nTF { \c@jCol }
5154   { \@@_error:nn { in~first~col } { \Ldots } }
5155   {
5156     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5157     { \@@_error:nn { in~last~col } { \Ldots } }
5158     {
5159       \@@_instruction_of_type:nnn { \c_false_bool } { Ldots }
5160       { #1 , down = #2 , up = #3 , middle = #4 }
5161     }
5162   }
5163   \bool_if:NF \l_@@_nullify_dots_bool
5164   { \phantom { \ensuremath { \@@_old_ldots: } } }
5165   \bool_gset_true:N \g_@@_empty_cell_bool
5166 }
```

```
5167 \cs_new_protected:Npn \@@_Cdots:
5168 { \@@_collect_options:n { \@@_Cdots_i } }
5169 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5170 {
5171   \int_if_zero:nTF { \c@jCol }
5172   { \@@_error:nn { in~first~col } { \Cdots } }
5173   {
5174     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5175     { \@@_error:nn { in~last~col } { \Cdots } }
5176     {
5177       \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5178       { #1 , down = #2 , up = #3 , middle = #4 }
5179     }
5180   }
5181   \bool_if:NF \l_@@_nullify_dots_bool
5182   { \phantom { \ensuremath { \@@_old_cdots: } } }
5183   \bool_gset_true:N \g_@@_empty_cell_bool
5184 }
```

```
5185 \cs_new_protected:Npn \@@_Vdots:
5186 { \@@_collect_options:n { \@@_Vdots_i } }
5187 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5188 {
5189   \int_if_zero:nTF { \c@iRow }
```

```

5190     { \@@_error:nn { in~first~row } { \Vdots } }
5191   {
5192     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5193       { \@@_error:nn { in~last~row } { \Vdots } }
5194     {
5195       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5196       { #1 , down = #2 , up = #3 , middle = #4 }
5197     }
5198   }
5199   \bool_if:NF \l_@@_nullify_dots_bool
5200   { \phantom { \ensuremath { \@@_old_vdots: } } } }
5201   \bool_gset_true:N \g_@@_empty_cell_bool
5202 }

5203 \cs_new_protected:Npn \@@_Ddots:
5204 { \@@_collect_options:n { \@@_Ddots_i } }
5205 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5206 {
5207   \int_case:nnF \c@iRow
5208   {
5209     0 { \@@_error:nn { in~first~row } { \Ddots } }
5210     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5211   }
5212   {
5213     \int_case:nnF \c@jCol
5214     {
5215       0 { \@@_error:nn { in~first~col } { \Ddots } }
5216       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5217     }
5218     {
5219       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5220       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5221       { #1 , down = #2 , up = #3 , middle = #4 }
5222     }
5223   }
5224 }
5225 \bool_if:NF \l_@@_nullify_dots_bool
5226 { \phantom { \ensuremath { \@@_old_ddots: } } } }
5227 \bool_gset_true:N \g_@@_empty_cell_bool
5228 }

5229 \cs_new_protected:Npn \@@_Iddots:
5230 { \@@_collect_options:n { \@@_Iddots_i } }
5231 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5232 {
5233   \int_case:nnF \c@iRow
5234   {
5235     0 { \@@_error:nn { in~first~row } { \Iddots } }
5236     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5237   }
5238   {
5239     \int_case:nnF \c@jCol
5240     {
5241       0 { \@@_error:nn { in~first~col } { \Iddots } }
5242       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5243     }
5244     {
5245       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5246       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5247       { #1 , down = #2 , up = #3 , middle = #4 }
5248     }
5249   }

```

```

5250     \bool_if:NF \l_@@_nullify_dots_bool
5251     { \phantom { \ensuremath { \@@_old_iddots: } } }
5252     \bool_gset_true:N \g_@@_empty_cell_bool
5253   }
5254 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5255 \keys_define:nn { nicematrix / Ddots }
5256 {
5257   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5258   draw-first .default:n = true ,
5259   draw-first .value_forbidden:n = true
5260 }

```

The command \@@\_Hspace: will be linked to \hspace in {NiceArray}.

```

5261 \cs_new_protected:Npn \@@_Hspace:
5262 {
5263   \bool_gset_true:N \g_@@_empty_cell_bool
5264   \hspace
5265 }

```

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`.

```

5266 \cs_set_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`.

```

5267 \cs_new:Npn \@@_Hdotsfor:
5268 {
5269   \bool_lazy_and:nnTF
5270     { \int_if_zero_p:n { \c@jCol } }
5271     { \int_if_zero_p:n { \l_@@_first_col_int } }
5272   {
5273     \bool_if:NTF \g_@@_after_col_zero_bool
5274     {
5275       \multicolumn { 1 } { c } { }
5276       \@@_Hdotsfor_i:
5277     }
5278     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5279   }
5280   {
5281     \multicolumn { 1 } { c } { }
5282     \@@_Hdotsfor_i:
5283   }
5284 }

```

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:`).

```

5285 \hook_gput_code:nnn { begindocument } { . }
5286 {

```

We don't put ! before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5287   \cs_new_protected:Npn \@@_Hdotsfor_i:
5288   { \@@_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`).

```

5289 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5290 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5291 {
5292   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5293   {
5294     \@@_Hdotsfor:nnnn
5295     { \int_use:N \c@iRow }
5296     { \int_use:N \c@jCol }
5297     { #2 }
5298     {
5299       #1 , #3 ,
5300       down = \exp_not:n { #4 } ,
5301       up = \exp_not:n { #5 } ,
5302       middle = \exp_not:n { #6 }
5303     }
5304   }
5305   \prg_replicate:nn { #2 - 1 }
5306   {
5307     &
5308     \multicolumn { 1 } { c } { }
5309     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5310   }
5311 }
5312 }

```

```

5313 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5314 {
5315   \bool_set_false:N \l_@@_initial_open_bool
5316   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5317 \int_set:Nn \l_@@_initial_i_int { #1 }
5318 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5319 \int_compare:nNnTF { #2 } = { \c_one_int }
5320 {
5321   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5322   \bool_set_true:N \l_@@_initial_open_bool
5323 }
5324 {
5325   \cs_if_exist:cTF
5326   {
5327     pgf @ sh @ ns @ \@@_env:
5328     - \int_use:N \l_@@_initial_i_int
5329     - \int_eval:n { #2 - 1 }
5330   }
5331   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5332   {
5333     \int_set:Nn \l_@@_initial_j_int { #2 }
5334     \bool_set_true:N \l_@@_initial_open_bool
5335   }
5336 }
5337 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5338 {
5339   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5340   \bool_set_true:N \l_@@_final_open_bool
5341 }
5342 {
5343   \cs_if_exist:cTF
5344   {
5345     pgf @ sh @ ns @ \@@_env:

```

```

5346         - \int_use:N \l_@@_final_i_int
5347         - \int_eval:n { #2 + #3 }
5348     }
5349     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5350     {
5351         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5352         \bool_set_true:N \l_@@_final_open_bool
5353     }
5354 }
5355 \bool_if:NT \g_@@_aux_found_bool
5356 {
5357     \group_begin:
5358     \@@_open_shorten:
5359     \int_if_zero:nTF { #1 }
5360     { \color { nicematrix-first-row } }
5361     {
5362         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5363         { \color { nicematrix-last-row } }
5364     }
5365     \keys_set:nn { nicematrix / xdots } { #4 }
5366     \@@_color:o \l_@@_xdots_color_tl
5367     \@@_actually_draw_Ldots:
5368     \group_end:
5369 }

```

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).

```

5370 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5371 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5372 }

```

```

5373 \hook_gput_code:nnn { begindocument } { . }
5374 {
5375     \cs_new_protected:Npn \@@_Vdotsfor:
5376     { \@@_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`).

```

5377 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5378 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5379 {
5380     \bool_gset_true:N \g_@@_empty_cell_bool
5381     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5382     {
5383         \@@_Vdotsfor:nnnn
5384         { \int_use:N \c_iRow }
5385         { \int_use:N \c_jCol }
5386         { #2 }
5387         {
5388             #1 , #3 ,
5389             down = \exp_not:n { #4 } ,
5390             up = \exp_not:n { #5 } ,
5391             middle = \exp_not:n { #6 }
5392         }
5393     }
5394 }
5395 }

```

**#1** is the number of row;

**#2** is the number of column;

#3 is the numbers of rows which are involved;

```

5396 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5397 {
5398   \bool_set_false:N \l_@@_initial_open_bool
5399   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5400   \int_set:Nn \l_@@_initial_j_int { #2 }
5401   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5402   \int_compare:nNnTF { #1 } = { \c_one_int }
5403   {
5404     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5405     \bool_set_true:N \l_@@_initial_open_bool
5406   }
5407   {
5408     \cs_if_exist:cTF
5409     {
5410       pgf @ sh @ ns @ \@@_env:
5411       - \int_eval:n { #1 - 1 }
5412       - \int_use:N \l_@@_initial_j_int
5413     }
5414     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5415     {
5416       \int_set:Nn \l_@@_initial_i_int { #1 }
5417       \bool_set_true:N \l_@@_initial_open_bool
5418     }
5419   }
5420   \int_compare:nNnTF { #1 + #3 - 1 } = { \c_iRow }
5421   {
5422     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5423     \bool_set_true:N \l_@@_final_open_bool
5424   }
5425   {
5426     \cs_if_exist:cTF
5427     {
5428       pgf @ sh @ ns @ \@@_env:
5429       - \int_eval:n { #1 + #3 }
5430       - \int_use:N \l_@@_final_j_int
5431     }
5432     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5433     {
5434       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5435       \bool_set_true:N \l_@@_final_open_bool
5436     }
5437   }
5438   \bool_if:NT \g_@@_aux_found_bool
5439   {
5440     \group_begin:
5441     \@@_open_shorten:
5442     \int_if_zero:nTF { #2 }
5443     { \color { nicematrix-first-col } }
5444     {
5445       \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5446       { \color { nicematrix-last-col } }
5447     }
5448     \keys_set:nn { nicematrix / xdots } { #4 }
5449     \@@_color:o \l_@@_xdots_color_tl
5450     \bool_if:NTF \l_@@_Vbrace_bool
5451     { \@@_actually_draw_Vbrace: }
5452     { \@@_actually_draw_Vdots: }
5453     \group_end:
5454   }

```

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).

```

5455 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5456 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5457 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5458 \NewDocumentCommand \@@_rotate: { 0 { } }
5459 {
5460   \bool_gset_true:N \g_@@_rotate_bool
5461   \keys_set:nn { nicematrix / rotate } { #1 }
5462   \ignorespaces
5463 }

5464 \keys_define:nn { nicematrix / rotate }
5465 {
5466   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5467   c .value_forbidden:n = true ,
5468   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5469 }

```

## 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>

```

5470 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5471 {
5472   \tl_if_empty:nTF { #2 }
5473   { #1 }
5474   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5475 }
5476 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5477 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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).

```

5478 \hook_gput_code:nnn { begindocument } { . }
5479 {

```

---

<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`.

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that’s why we are in a `\AtBeginDocument`).

```

5480 \tl_set_rescan:Nnn \l_tmpa_tl { }
5481 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5482 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5483 {
5484   \group_begin:
5485   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5486   \@@_color:o \l_@@_xdots_color_tl
5487   \use:e
5488   {
5489     \@@_line_i:nn
5490     { \@@_double_int_eval:n #2 - \q_stop }
5491     { \@@_double_int_eval:n #3 - \q_stop }
5492   }
5493   \group_end:
5494 }
5495 }

5496 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5497 {
5498   \bool_set_false:N \l_@@_initial_open_bool
5499   \bool_set_false:N \l_@@_final_open_bool
5500   \bool_lazy_or:nnTF
5501   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5502   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5503   { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5504 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5505 }

5506 \hook_gput_code:nnn { begindocument } { . }
5507 {
5508   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5509   {

```

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:`.

```

5510 \c_@@_pgfortikzpicture_tl
5511 \@@_draw_line_iii:nn { #1 } { #2 }
5512 \c_@@_endpgfortikzpicture_tl
5513 }
5514 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5515 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5516 {
5517   \pgfrememberpicturepositiononpagetrue
5518   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5519   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5520   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5521   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5522   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5523   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5524   \@@_draw_line:
5525 }

```

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.

```
5526 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNtT { \c@iRow } < }
5527 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNfT { \c@jCol } < }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5528 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5529 {
5530   \tl_gput_right:Ne \g_@@_row_style_tl
5531   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5532   \exp_not:N
5533   \@@_if_row_less_than:nn
5534   { \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`).

```
5535   {
5536     \exp_not:N
5537     \@@_if_col_greater_than:nn
5538     { \int_eval:n { \c@jCol } }
5539     { \exp_not:n { #1 } \scan_stop: }
5540   }
5541 }
5542 }
5543 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5544 \keys_define:nn { nicematrix / RowStyle }
5545 {
5546   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5547   cell-space-top-limit .value_required:n = true ,
5548   cell-space-top-limit+ .code:n =
5549     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5550   cell-space-top-limit+ .value_required:n = true ,
5551   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5552   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5553   cell-space-bottom-limit .value_required:n = true ,
5554   cell-space-bottom-limit+ .code:n =
5555     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5556   cell-space-bottom-limit+ .value_required:n = true ,
5557   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5558   cell-space-limits .meta:n =
5559   {
5560     cell-space-top-limit = #1 ,
5561     cell-space-bottom-limit = #1 ,
5562   } ,
5563   cell-space-limits+ .meta:n =
5564   {
```

```

5565         cell-space-top-limit += #1 ,
5566         cell-space-bottom-limit += #1 ,
5567     } ,
5568     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5569     color .tl_set:N = \l_@@_color_tl ,
5570     color .value_required:n = true ,
5571     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5572     bold .default:n = true ,
5573     nb-rows .code:n =
5574         \str_if_eq:eeTF { #1 } { * }
5575         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5576         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5577     nb-rows .value_required:n = true ,
5578     fill .tl_set:N = \l_@@_fill_tl ,
5579     fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

5580     opacity .tl_set:N = \l_@@_opacity_tl ,
5581     opacity .value_required:n = true ,
5582     rowcolor .tl_set:N = \l_@@_fill_tl ,
5583     rowcolor .value_required:n = true ,
5584     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5585     rounded-corners .default:n = 4 pt ,
5586     unknown .code:n =
5587         \@@_unknown_key:nn
5588         { nicematrix / RowStyle }
5589         { Unknown-key~for~RowStyle }
5590 }

```

```

5591 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5592 {
5593     \group_begin:
5594     \tl_clear:N \l_@@_fill_tl
5595     \tl_clear:N \l_@@_opacity_tl
5596     \tl_clear:N \l_@@_color_tl
5597     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5598     \dim_zero:N \l_@@_rounded_corners_dim
5599     \dim_zero:N \l_tmpa_dim
5600     \dim_zero:N \l_tmpb_dim
5601     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `fill` (or its alias `rowcolor`) has been used.

```

5602     \tl_if_empty:NF \l_@@_fill_tl
5603     {
5604         \@@_add_opacity_to_fill:
5605         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5606         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5607         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5608         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5609         {
5610             \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5611             - *
5612         }
5613         { \dim_use:N \l_@@_rounded_corners_dim }
5614     }
5615 }
5616 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5617     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5618     {
5619         \@@_put_in_row_style:e

```

```

5620     {
5621         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5622     }

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5623         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5624         { \dim_use:N \l_tmpa_dim }
5625     }
5626 }
5627 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5628 \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5629 {
5630     \@@_put_in_row_style:e
5631     {
5632         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5633         {
5634             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5635             { \dim_use:N \l_tmpb_dim }
5636         }
5637     }
5638 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5639 \tl_if_empty:NF \l_@@_color_tl
5640 {
5641     \@@_put_in_row_style:e
5642     {
5643         \mode_leave_vertical:
5644         \@@_color:n { \l_@@_color_tl }
5645     }
5646 }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5647 \bool_if:NT \l_@@_bold_row_style_bool
5648 {
5649     \@@_put_in_row_style:n
5650     {
5651         \exp_not:n
5652         {
5653             \if_mode_math:
5654             $ % $
5655             \bfseries \boldmath
5656             $ % $
5657             \else:
5658             \bfseries \boldmath
5659             \fi:
5660         }
5661     }
5662 }
5663 \group_end:
5664 \g_@@_row_style_tl
5665 \ignorespaces
5666 }

```

The following commande must *not* be protected.

```

5667 \cs_new:Npn \@@_rounded_from_row:n #1
5668 {
5669     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5670     { \int_eval:n { #1 } - 1 }
5671     {
5672         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }

```

```

5673     - \exp_not:n { \int_use:N \c@jCol }
5674   }
5675   { \dim_use:N \l_@@_rounded_corners_dim }
5676 }

```

## 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).

```

5677 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5678 {

```

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.

```

5679   \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`.

```

5680   \str_if_in:nnF { #1 } { !! }
5681   {
5682     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5683     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5684   }
5685   \int_if_zero:nTF { \l_tmpa_int }

```

First, the case where the color is a *new* color (not in the sequence).

```

5686   {
5687     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5688     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5689   }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5690   { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5691 }
5692 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5693 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

The following command must be used within a `\pgfpicture`.

```

5694 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5695 {
5696   \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5697   {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5698   \group_begin:
5699   \pgfsetcornersarced
5700   {
5701     \pgfpoint
5702     { \l_@@_tab_rounded_corners_dim }
5703     { \l_@@_tab_rounded_corners_dim }
5704   }

```

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.

```

5705   \bool_if:NTF \l_@@_hvlines_bool
5706   {
5707     \pgfpathrectanglecorners
5708     {
5709       \pgfpointadd
5710       { \@@_qpoint:n { row-1 } }
5711       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5712     }
5713     {
5714       \pgfpointadd
5715       {
5716         \@@_qpoint:n
5717         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5718       }
5719       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5720     }
5721   }
5722   {
5723     \pgfpathrectanglecorners
5724     { \@@_qpoint:n { row-1 } }
5725     {
5726       \pgfpointadd
5727       {
5728         \@@_qpoint:n
5729         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5730       }
5731       { \pgfpoint \c_zero_dim \arrayrulewidth }
5732     }
5733   }
5734   \pgfusepath { clip }
5735   \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5736   }
5737 }

```

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-t1`).

```

5738 \cs_new_protected:Npn \@@_actually_color:
5739 {
5740   \pgfpicture
5741   \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.

```

5742 \@@_clip_with_rounded_corners:
5743 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5744 {
5745   \int_compare:nNnTF { ##1 } = { \c_one_int }
5746   {
5747     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5748     \use:c { g_@@_color _ 1 _tl }
5749     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5750   }
5751   {
5752     \begin { pgfscope }
5753       \@@_color_opacity: ##2
5754       \use:c { g_@@_color _ ##1 _tl }
5755       \tl_gclear:c { g_@@_color _ ##1 _tl }
5756       \pgfusepath { fill }
5757     \end { pgfscope }
5758   }
5759 }
5760 \endpgfpicture
5761 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5762 \cs_new_protected:Npn \@@_color_opacity:
5763 {
5764   \peek_meaning:NTF [
5765     { \@@_color_opacity:w }
5766     { \@@_color_opacity:w [ ] }
5767   }

```

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.

```

5768 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5769 {
5770   \tl_clear:N \l_tmpa_tl
5771   \keys_set_known:nnN { 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.

```

5772   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5773   \tl_if_empty:NTF \l_tmpb_tl
5774     { \@declaredcolor }
5775     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5776 }

```

The following set of keys is used by the command `\@@_color_opacity:w`.

```

5777 \keys_define:nn { nicematrix / color-opacity }
5778 {
5779   opacity .tl_set:N          = \l_tmpa_tl ,
5780   opacity .value_required:n = true
5781 }

```

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

```

5782 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5783 {
5784   \def \l_@@_rows_tl { #1 }
5785   \def \l_@@_cols_tl { #2 }
5786   \@@_cartesian_path:
5787 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
5788 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5789 {
5790   \tl_if_blank:nF { #2 }
5791   {
5792     \@@_add_to_colors_seq:en
5793     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5794     { \@@_cartesian_color:nn { #3 } { - } }
5795   }
5796 }
```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
5797 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5798 {
5799   \tl_if_blank:nF { #2 }
5800   {
5801     \@@_add_to_colors_seq:en
5802     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5803     { \@@_cartesian_color:nn { - } { #3 } }
5804   }
5805 }
```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
5806 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5807 {
5808   \tl_if_blank:nF { #2 }
5809   {
5810     \@@_add_to_colors_seq:en
5811     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5812     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5813   }
5814 }
```

The last argument is the radius of the corners of the rectangle.

```
5815 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5816 {
5817   \tl_if_blank:nF { #2 }
5818   {
5819     \@@_add_to_colors_seq:en
5820     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5821     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5822   }
5823 }
```

The last argument is the radius of the corners of the rectangle.

```
5824 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5825 {
5826   \@@_cut_on_hyphen:w #1 \q_stop
5827   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5828   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5829   \@@_cut_on_hyphen:w #2 \q_stop
5830   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5831   \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`.

```
5832 \@@_cartesian_path:n { #3 }
5833 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5834 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5835 {
5836   \clist_map_inline:nn { #3 }
5837     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5838 }

5839 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5840 {
5841   \int_step_inline:nn { \c@iRow }
5842   {
5843     \int_step_inline:nn { \c@jCol }
5844     {
5845       \int_if_even:nTF { #####1 + ##1 }
5846         { \@@_cellcolor [ #1 ] { #2 } }
5847         { \@@_cellcolor [ #1 ] { #3 } }
5848       { ##1 - #####1 }
5849     }
5850   }
5851 }

```

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”.

```

5852 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5853 {
5854   \@@_rectanglecolor [ #1 ] { #2 }
5855   { 1 - 1 }
5856   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5857 }

5858 \keys_define:nn { nicematrix / rowcolors }
5859 {
5860   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5861   respect-blocks .default:n = true ,
5862   cols .tl_set:N = \l_@@_cols_tl ,
5863   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5864   restart .default:n = true ,
5865   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5866 }

```

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*.

```

5867 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5868 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5869   \group_begin:
5870   \seq_clear_new:N \l_@@_colors_seq
5871   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5872   \tl_clear_new:N \l_@@_cols_tl
5873   \tl_set:Nn \l_@@_cols_tl { - }
5874   \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).

```

5875 \int_zero_new:N \l_@@_color_int
5876 \int_set_eq:NN \l_@@_color_int \c_one_int
5877 \bool_if:NT \l_@@_respect_blocks_bool
5878 {

```

We don't want to take into account a block which is completely 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`).

```

5879 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5880 \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5881 { \@@_not_in_exterior_p:nnnnn ##1 }
5882 }
5883 \pgfpicture
5884 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5885 \clist_map_inline:nn { #2 }
5886 {
5887 \tl_set:Nn \l_tmpa_tl { ##1 }
5888 \tl_if_in:NnTF \l_tmpa_tl { - }
5889 { \@@_cut_on_hyphen:w ##1 \q_stop }
5890 { \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.

```

5891 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5892 \int_set:Nn \l_@@_color_int
5893 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5894 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5895 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5896 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5897 \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).

```

5898 \bool_if:NT \l_@@_respect_blocks_bool
5899 {
5900 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5901 { \@@_intersect_our_row_p:nnnnn ###1 }
5902 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5903 }
5904 \tl_set:Nn \l_@@_rows_tl
5905 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5906 \tl_set:Nn \l_@@_color_tl
5907 {
5908 \@@_color_index:n
5909 {
5910 \int_mod:nn
5911 { \l_@@_color_int - 1 }
5912 { \seq_count:N \l_@@_colors_seq }
5913 + 1
5914 }
5915 }
5916 \tl_if_empty:NF \l_@@_color_tl
5917 {
5918 \@@_add_to_colors_seq:ee
5919 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5920 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5921 }
5922 \int_incr:N \l_@@_color_int

```

```

5923         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5924     }
5925 }
5926 \endpgfpicture
5927 \group_end:
5928 }

```

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.

```

5929 \cs_new:Npn \@@_color_index:n #1
5930 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5931     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5932     { \@@_color_index:n { #1 - 1 } }
5933     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5934 }

```

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 curryfication.

```

5935 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5936 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5937 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5938 {
5939     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5940     { \int_set:Nn \l_tmpb_int { #3 } }
5941 }

```

```

5942 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5943 {
5944     \int_if_zero:nTF { #4 }
5945     { \prg_return_false: }
5946     {
5947         \int_compare:nNnTF { #2 } > { \c@jCol }
5948         { \prg_return_false: }
5949         { \prg_return_true: }
5950     }
5951 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5952 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5953 {
5954     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5955     { \prg_return_false: }
5956     {
5957         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5958         { \prg_return_false: }
5959         { \prg_return_true: }
5960     }
5961 }

```

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`).

```

5962 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5963 {
5964   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5965   {
5966     \bool_if:NTF \l_@@_nocolor_used_bool
5967     { \@@_cartesian_path_normal_ii: }
5968     {
5969       \clist_if_empty:NTF \l_@@_corners_cells_clist
5970       { \@@_cartesian_path_normal_i:n { #1 } }
5971       { \@@_cartesian_path_normal_ii: }
5972     }
5973   }
5974   { \@@_cartesian_path_normal_i:n { #1 } }
5975 }

```

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.

```

5976 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5977 {
5978   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5979   \clist_map_inline:Nn \l_@@_cols_tl
5980   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5981     \def \l_tmpa_tl { ##1 }
5982     \tl_if_in:NnTF \l_tmpa_tl { - }
5983     { \@@_cut_on_hyphen:w ##1 \q_stop }
5984     { \def \l_tmpb_tl { ##1 } }
5985     \tl_if_empty:NTF \l_tmpa_tl
5986     { \def \l_tmpa_tl { 1 } }
5987     {
5988       \str_if_eq:eeT { \l_tmpa_tl } { * }
5989       { \def \l_tmpa_tl { 1 } }
5990     }
5991     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5992     { \@@_error:n { Invalid~col~number } }
5993     \tl_if_empty:NTF \l_tmpb_tl
5994     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5995     {
5996       \str_if_eq:eeT { \l_tmpb_tl } { * }
5997       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5998     }
5999     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
6000     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6001     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6002     \@@_qpoint:n { col - \l_tmpa_tl }
6003     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
6004     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6005     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6006     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6007     \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.

```

6008     \clist_map_inline:Nn \l_@@_rows_tl
6009     {
6010       \def \l_tmpa_tl { ####1 }
6011       \tl_if_in:NnTF \l_tmpa_tl { - }
6012       { \@@_cut_on_hyphen:w ####1 \q_stop }

```

```

6013     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6014 \tl_if_empty:NTF \l_tmpa_tl
6015     { \def \l_tmpa_tl { 1 } }
6016     {
6017         \str_if_eq:eeT { \l_tmpa_tl } { * }
6018         { \def \l_tmpa_tl { 1 } }
6019     }
6020 \tl_if_empty:NTF \l_tmpb_tl
6021     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6022     {
6023         \str_if_eq:eeT { \l_tmpb_tl } { * }
6024         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6025     }
6026 \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
6027     { \@@_error:n { Invalid~row~number } }
6028 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
6029     { \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`.

```

6030 \cs_if_exist:cF
6031     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6032     {
6033         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6034         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6035         \@@_qpoint:n { row - \l_tmpa_tl }
6036         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6037         \pgfpathrectanglecorners
6038             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6039             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6040     }
6041 }
6042 }
6043 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6044 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6045     {
6046         \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6047         \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6048 \clist_map_inline:Nn \l_@@_cols_tl
6049     {
6050         \@@_qpoint:n { col - ##1 }
6051         \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
6052             { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6053             { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6054         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6055         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6056 \clist_map_inline:Nn \l_@@_rows_tl
6057     {
6058         \@@_if_in_corner:nF { #####1 - ##1 }
6059         {
6060             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6061             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6062             \@@_qpoint:n { row - #####1 }
6063             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6064             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6065                 {
6066                     \pgfpathrectanglecorners
6067                         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6068                         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

```

6069     }
6070   }
6071 }
6072 }
6073 }

```

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`).

```

6074 \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

```

6075 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6076 {
6077   \bool_set_true:N \l_@@_nocolor_used_bool
6078   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6079   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6080   \clist_map_inline:Nn \l_@@_rows_tl
6081   {
6082     \clist_map_inline:Nn \l_@@_cols_tl
6083     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6084   }
6085 }

```

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`.

```

6086 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6087 {
6088   \clist_set_eq:NN \l_tmpa_clist #1
6089   \clist_clear:N #1
6090   \clist_map_inline:Nn \l_tmpa_clist
6091   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6092     \def \l_tmpa_tl { ##1 }
6093     \tl_if_in:NnTF \l_tmpa_tl { - }
6094     { \@@_cut_on_hyphen:w ##1 \q_stop }
6095     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6096     \bool_lazy_or:nnT
6097     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
6098     { \tl_if_blank_p:o \l_tmpa_tl }
6099     { \def \l_tmpa_tl { 1 } }
6100     \bool_lazy_or:nnT
6101     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
6102     { \tl_if_blank_p:o \l_tmpb_tl }
6103     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6104     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6105     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6106     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
6107     { \clist_put_right:Nn #1 { ####1 } }
6108   }
6109 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6110 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6111 {
6112   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6113   {

```

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).

```

6114     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6115     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6116   }
6117   \ignorespaces
6118 }

6119 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6120 { \@@_error:n { cellcolor~in~Block } }
6121 % \end{macrocode}
6122 %
6123 % \begin{macrocode}
6124 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6125 { \@@_error:n { rowcolor~in~Block } }
6126 % \end{macrocode}
6127 %
6128 % \bigskip
6129 % The following command will be linked to |\rowcolor| in the tabular.
6130 % \begin{macrocode}
6131 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6132 {
6133   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6134   {
6135     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6136     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6137     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6138   }
6139   \ignorespaces
6140 }

```

The following command will be linked to \rowcolors in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6141 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6142 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to \rowlistcolors in the tabular.

```

6143 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6144 {

```

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.

```

6145   \seq_gclear:N \g_tmpa_seq
6146   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6147   { \@@_rowlistcolors_tabular:nnnn #1 }
6148   \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.

```

6149   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6150   {
6151     { \int_use:N \c@iRow }
6152     { \exp_not:n { #1 } }
6153     { \exp_not:n { #2 } }
6154     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6155   }

```

```

6156 \ignorespaces
6157 }

```

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`).

```

6158 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6159 {
6160   \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).

```

6161   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6162   {
6163     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6164     {
6165       \@@_rowlistcolors
6166       [ \exp_not:n { #2 } ]
6167       { #1 - \int_eval:n { \c@iRow - 1 } }
6168       { \exp_not:n { #3 } }
6169       [ \exp_not:n { #4 } ]
6170     }
6171   }
6172 }

```

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.

```

6173 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6174 {
6175   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6176   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6177   \seq_gclear:N \g_@@_rowlistcolors_seq
6178 }

6179 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6180 {
6181   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6182   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6183 }

```

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.

```

6184 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6185 {

```

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).

```

6186   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6187   {

```

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).

```

6188   \tl_gput_left:Ne \g_@@_pre_code_before_tl

```

```

6189         {
6190             \exp_not:N \columncolor [ #1 ]
6191             { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6192         }
6193     }
6194 }

6195 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6196 {
6197     \clist_map_inline:nn { #1 }
6198     {
6199         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6200         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6201         \columncolor { nocolor } { ##1 }
6202     }
6203 }

6204 \cs_new_protected:Npn \@@_EmptyRow:n #1
6205 {
6206     \clist_map_inline:nn { #1 }
6207     {
6208         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6209         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6210         \rowcolor { nocolor } { ##1 }
6211     }
6212 }

```

## 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`.

```

6213 \cs_set_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`.

```

6214 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6215 {
6216     \int_if_zero:nTF { \l_@@_first_col_int }
6217     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6218     {
6219         \int_if_zero:nTF { \c@jCol }
6220         {
6221             \int_compare:nNnF { \c@iRow } = { -1 }
6222             {
6223                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6224                 { #1 }
6225             }
6226         }
6227         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6228     }
6229 }

```



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.

```

6230 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6231 {
6232   \int_if_zero:nF { \c@iRow }
6233   {
6234     \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6235     {
6236       \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6237       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6238     }
6239   }
6240 }

```

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`.

```

6241 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6242 {
6243   \IfPackageLoadedTF { tikz }
6244   {
6245     \IfPackageLoadedTF { booktabs }
6246     { #2 }
6247     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6248   }
6249   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6250 }
6251 \NewExpandableDocumentCommand { \@@_TopRule } { }
6252 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6253 \cs_new:Npn \@@_TopRule_i:
6254 {
6255   \noalign \bgroup
6256   \peek_meaning:NTF [
6257   { \@@_TopRule_ii: }
6258   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6259 }
6260 \NewDocumentCommand \@@_TopRule_ii: { o }
6261 {
6262   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6263   {
6264     \@@_hline:n
6265     {
6266       position = \int_eval:n { \c@iRow + 1 } ,
6267       tikz =
6268       {
6269         line-width = #1 ,
6270         yshift = 0.25 \arrayrulewidth ,
6271         shorten-< = - 0.5 \arrayrulewidth
6272       } ,
6273       total-width = #1
6274     }
6275   }
6276   \skip_vertical:n { \belowrulesep + #1 }
6277   \egroup
6278 }
6279 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6280 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

```

```

6281 \cs_new:Npn \@@_BottomRule_i:
6282 {
6283   \noalign \bgroup
6284   \peek_meaning:NTF [
6285     { \@@_BottomRule_ii: }
6286     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6287   }
6288 \NewDocumentCommand \@@_BottomRule_ii: { o }
6289 {
6290   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6291   {
6292     \@@_hline:n
6293     {
6294       position = \int_eval:n { \c@iRow + 1 } ,
6295       tikz =
6296       {
6297         line-width = #1 ,
6298         yshift = 0.25 \arrayrulewidth ,
6299         shorten~< = - 0.5 \arrayrulewidth
6300       } ,
6301       total-width = #1 ,
6302     }
6303   }
6304   \skip_vertical:N \aboverulesep
6305   \@@_create_row_node_i:
6306   \skip_vertical:n { #1 }
6307   \egroup
6308 }
6309 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6310 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6311 \cs_new:Npn \@@_MidRule_i:
6312 {
6313   \noalign \bgroup
6314   \peek_meaning:NTF [
6315     { \@@_MidRule_ii: }
6316     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6317   }
6318 \NewDocumentCommand \@@_MidRule_ii: { o }
6319 {
6320   \skip_vertical:N \aboverulesep
6321   \@@_create_row_node_i:
6322   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6323   {
6324     \@@_hline:n
6325     {
6326       position = \int_eval:n { \c@iRow + 1 } ,
6327       tikz =
6328       {
6329         line-width = #1 ,
6330         yshift = 0.25 \arrayrulewidth ,
6331         shorten~< = - 0.5 \arrayrulewidth
6332       } ,
6333       total-width = #1 ,
6334     }
6335   }
6336   \skip_vertical:n { \belowrulesep + #1 }
6337   \egroup
6338 }

```

## 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.

```

6339 \keys_define:nn { nicematrix / Rules }
6340 {
6341   position .int_set:N = \l_@@_position_int ,
6342   position .value_required:n = true ,
6343   start .int_set:N = \l_@@_start_int ,
6344   end .code:n =
6345     \bool_lazy_or:nnTF
6346     { \tl_if_empty_p:n { #1 } }
6347     { \str_if_eq_p:ee { #1 } { last } }
6348     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6349     { \int_set:Nn \l_@@_end_int { #1 } }
6350 }

```

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.

```

6351 \keys_define:nn { nicematrix / RulesBis }
6352 {
6353   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6354   multiplicity .initial:n = 1 ,
6355   dotted .bool_set:N = \l_@@_dotted_bool ,
6356   dotted .initial:n = false ,
6357   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]`.

```

6358   color .code:n =
6359     \@@_set_CTarc:n { #1 }
6360     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6361   color .value_required:n = true ,
6362   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6363   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.

```

6364   tikz .code:n =
6365     \IfPackageLoadedTF { tikz }
6366     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6367     { \@@_error:n { tikz~without~tikz } } ,
6368   tikz .value_required:n = true ,
6369   total-width .dim_set:N = \l_@@_rule_width_dim ,
6370   total-width .value_required:n = true ,
6371   width .meta:n = { total-width = #1 } ,
6372   unknown .code:n =
6373     \@@_unknown_key:nn
6374     { nicematrix / RulesBis }
6375     { Unknown-key-for-RulesBis }
6376 }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
6377 \cs_new_protected:Npn \@@_vline:n #1
6378 {
```

The group is for the options.

```
6379 \group_begin:
6380 \int_set_eq:NN \l_@@_end_int \c@iRow
6381 \keys_set_known:nN { 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).

```
6382 \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6383 \@@_vline_i:
6384 \group_end:
6385 }
```

```
6386 \cs_new_protected:Npn \@@_vline_i:
6387 {
```

`\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`.

```
6388 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6389 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6390 \l_tmpa_tl
6391 {
```

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.

```
6392 \bool_gset_true:N \g_tmpa_bool
6393 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6394 { \@@_test_vline_in_block:nnnnn ##1 }
6395 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6396 { \@@_test_vline_in_block:nnnnn ##1 }
6397 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6398 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6399 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6400 \bool_if:NTF \g_tmpa_bool
6401 {
6402 \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.

```
6403 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6404 }
6405 {
6406 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6407 {
6408 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6409 \@@_vline_ii:
6410 \int_zero:N \l_@@_local_start_int
6411 }
6412 }
6413 }
6414 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6415 {
6416 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6417 \@@_vline_ii:
6418 }
6419 }
```

```

6420 \cs_new_protected:Npn \@@_test_in_corner_v:
6421 {
6422   \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6423   {
6424     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6425     { \bool_set_false:N \g_tmpa_bool }
6426   }
6427   {
6428     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6429     {
6430       \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6431       { \bool_set_false:N \g_tmpa_bool }
6432       {
6433         \@@_if_in_corner:nT
6434         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6435         { \bool_set_false:N \g_tmpa_bool }
6436       }
6437     }
6438   }
6439 }

6440 \cs_new_protected:Npn \@@_vline_ii:
6441 {
6442   \tl_clear:N \l_@@_tikz_rule_tl
6443   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6444   \bool_if:NTF \l_@@_dotted_bool
6445   { \@@_vline_iv: }
6446   {
6447     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6448     { \@@_vline_iii: }
6449     { \@@_vline_v: }
6450   }
6451 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6452 \cs_new_protected:Npn \@@_vline_iii:
6453 {
6454   \pgfpicture
6455   \pgfrememberpicturepositiononpagetrue
6456   \pgf@relevantforpicturesizefalse
6457   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6458   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6459   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6460   \dim_set:Nn \l_tmpb_dim
6461   {
6462     \pgf@x
6463     - 0.5 \l_@@_rule_width_dim
6464     +
6465     ( \arrayrulewidth * \l_@@_multiplicity_int
6466       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6467   }
6468   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6469   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6470   \bool_lazy_all:nT
6471   {
6472     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6473     { \cs_if_exist_p:N \CT@drsc@ }
6474     { ! \tl_if_blank_p:o \CT@drsc@ }
6475   }
6476   {
6477     \group_begin:
6478     \CT@drsc@

```

```

6479 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6480 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6481 \dim_set:Nn \l_@@_tmpd_dim
6482 {
6483   \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6484   * ( \l_@@_multiplicity_int - 1 )
6485 }
6486 \pgfpathrectanglecorners
6487 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6488 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6489 \pgfusepath { fill }
6490 \group_end:
6491 }
6492 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6493 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6494 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6495 {
6496   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6497   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6498   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6499 }
6500 \CT@arc@
6501 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6502 \pgfsetrectcap
6503 \pgfusepathqstroke
6504 \endpgfpicture
6505 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6506 \cs_new_protected:Npn \@@_vline_iv:
6507 {
6508   \pgfpicture
6509   \pgfrememberpicturepositiononpagetrue
6510   \pgf@relevantforpicturesizefalse
6511   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6512   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6513   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6514   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6515   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6516   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6517   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6518   \CT@arc@
6519   \@@_draw_line:
6520   \endpgfpicture
6521 }

```

The following code is for the case when the user uses the key `tikz`.

```

6522 \cs_new_protected:Npn \@@_vline_v:
6523 {
6524   \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).

```

6525   \CT@arc@
6526   \tl_if_empty:NF \l_@@_rule_color_tl
6527   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6528   \pgfrememberpicturepositiononpagetrue
6529   \pgf@relevantforpicturesizefalse
6530   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6531   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6532   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6533   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }

```

```

6534 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6535 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6536 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6537 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6538 ( \l_tmpb_dim , \l_tmpa_dim ) --
6539 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6540 \end { tikzpicture }
6541 }

```

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).

```

6542 \cs_new_protected:Npn \@@_draw_vlines:
6543 {
6544   \int_step_inline:nnn
6545   {
6546     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6547     { 2 }
6548     { 1 }
6549   }
6550   {
6551     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6552     { \c@jCol }
6553     { \int_eval:n { \c@jCol + 1 } } }
6554 }
6555 {
6556   \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6557   { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6558   { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6559 }
6560 }

```

## 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}`.

```

6561 \cs_new_protected:Npn \@@_hline:n #1
6562 {

```

The group is for the options.

```

6563   \group_begin:
6564   \int_set_eq:NN \l_@@_end_int \c@jCol
6565   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6566   \@@_hline_i:
6567   \group_end:
6568 }

```

```

6569 \cs_new_protected:Npn \@@_hline_i:
6570 {

```

`\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`.

```

6571   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6572   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6573   \l_tmpb_tl
6574   {

```

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.

```

6575   \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6576 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6577 { \@@_test_hline_in_block:nnnnn ##1 }

6578 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6579 { \@@_test_hline_in_block:nnnnn ##1 }
6580 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6581 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6582 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6583 \bool_if:NTF \g_tmpa_bool
6584 {
6585     \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.

```

6586     { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6587 }
6588 {
6589     \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6590     {
6591         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6592         \@@_hline_ii:
6593         \int_zero:N \l_@@_local_start_int
6594     }
6595 }
6596 }
6597 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6598 {
6599     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6600     \@@_hline_ii:
6601 }
6602 }

```

```

6603 \cs_new_protected:Npn \@@_test_in_corner_h:
6604 {
6605     \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6606     {
6607         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6608         { \bool_set_false:N \g_tmpa_bool }
6609     }
6610     {
6611         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6612         {
6613             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6614             { \bool_set_false:N \g_tmpa_bool }
6615             {
6616                 \@@_if_in_corner:nT
6617                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6618                 { \bool_set_false:N \g_tmpa_bool }
6619             }
6620         }
6621     }
6622 }

```

```

6623 \cs_new_protected:Npn \@@_hline_ii:
6624 {
6625     \tl_clear:N \l_@@_tikz_rule_tl
6626     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6627     \bool_if:NTF \l_@@_dotted_bool
6628     { \@@_hline_iv: }
6629     {
6630         \tl_if_empty:NTF \l_@@_tikz_rule_tl

```



```

6631         { \@@_hline_iii: }
6632         { \@@_hline_v: }
6633     }
6634 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6635 \cs_new_protected:Npn \@@_hline_iii:
6636 {
6637     \pgfpicture
6638     \pgfrememberpicturepositiononpagetrue
6639     \pgf@relevantforpicturesizefalse
6640     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6641     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6642     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6643     \dim_set:Nn \l_tmpb_dim
6644     {
6645         \pgf@y
6646         - 0.5 \l_@@_rule_width_dim
6647         +
6648         ( \arrayrulewidth * \l_@@_multiplicity_int
6649           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6650     }
6651     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6652     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6653     \bool_lazy_all:nT
6654     {
6655         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6656         { \cs_if_exist_p:N \CT@drsc@ }
6657         { ! \tl_if_blank_p:o \CT@drsc@ }
6658     }
6659     {
6660         \group_begin:
6661         \CT@drsc@
6662         \dim_set:Nn \l_@@_tmpd_dim
6663         {
6664             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6665             * ( \l_@@_multiplicity_int - 1 )
6666         }
6667         \pgfpathrectanglecorners
6668         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6669         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6670         \pgfusepathqfill
6671         \group_end:
6672     }
6673     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6674     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6675     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6676     {
6677         \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6678         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6679         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6680     }
6681     \CT@arc@
6682     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6683     \pgfsetrectcap
6684     \pgfusepathqstroke
6685     \endpgfpicture
6686 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6687 \cs_new_protected:Npn \@@_hline_iv:
6688 {
6689   \pgfpicture
6690   \pgfrememberpicturepositiononpagetrue
6691   \pgf@relevantforpicturesizefalse
6692   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6693   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6694   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6695   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6696   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6697   \int_compare:nNt { \l_@@_local_start_int } = { \c_one_int }
6698   {
6699     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6700     \bool_if:NF \g_@@_delims_bool
6701     { \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.

```

6702     \tl_if_eq:NnF \g_@@_left_delim_tl (
6703       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6704     )
6705     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6706     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6707     \int_compare:nNt { \l_@@_local_end_int } = { \c_jCol }
6708     {
6709       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6710       \bool_if:NF \g_@@_delims_bool
6711       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6712       \tl_if_eq:NnF \g_@@_right_delim_tl (
6713         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6714       )
6715       \CT@arc@
6716       \@@_draw_line:
6717       \endpgfpicture
6718     }

```

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`).

```

6719 \cs_new_protected:Npn \@@_hline_v:
6720 {
6721   \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).

```

6722   \CT@arc@
6723   \tl_if_empty:NF \l_@@_rule_color_tl

```

```

6724     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6725     \pgfrememberpicturepositiononpagetrue
6726     \pgf@relevantforpicturesizefalse
6727     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6728     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6729     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6730     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6731     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6732     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6733     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6734     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6735     ( \l_tmpa_dim , \l_tmpb_dim ) --
6736     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6737     \end { tikzpicture }
6738 }

```

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).

```

6739 \cs_new_protected:Npn \@@_draw_hlines:
6740 {
6741   \int_step_inline:nnn
6742   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6743   {
6744     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6745     { \c@iRow }
6746     { \int_eval:n { \c@iRow + 1 } }
6747   }
6748   {
6749     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6750     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6751     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6752   }
6753 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6754 \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.

```

6755 \cs_set:Npn \@@_Hline_i:n #1
6756 {
6757   \peek_remove_spaces:n
6758   {
6759     \peek_meaning:NTF \Hline
6760     { \@@_Hline_ii:nn { #1 + 1 } }
6761     { \@@_Hline_iii:n { #1 } }
6762   }
6763 }
6764 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6765 \cs_set:Npn \@@_Hline_iii:n #1
6766 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6767 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6768 {
6769   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6770   \skip_vertical:N \l_@@_rule_width_dim
6771   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6772   {
6773     \@@_hline:n
6774     {
6775       multiplicity = #1 ,
6776       position = \int_eval:n { \c@iRow + 1 } ,

```

```

6777         total-width = \dim_use:N \l_@@_rule_width_dim ,
6778         #2
6779     }
6780 }
6781 \egroup
6782 }

```

### 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`).

```

6783 \cs_new_protected:Npn \@@_custom_line:n #1
6784 {
6785     \str_clear_new:N \l_@@_command_str
6786     \str_clear_new:N \l_@@_ccommand_str
6787     \str_clear_new:N \l_@@_letter_str
6788     \tl_clear_new:N \l_@@_other_keys_tl
6789     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6790     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6791     {
6792         \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6793         \str_set:Ne \l_@@_command_str
6794         { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6795     }
6796     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_cccommand_str }
6797     {
6798         \str_set:Ne \l_@@_cccommand_str
6799         { \str_tail:N \l_@@_cccommand_str }
6800         \str_set:Ne \l_@@_cccommand_str
6801         { \str_range:Nnn \l_@@_cccommand_str { 1 } { -2 } }
6802     }

```

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.

```

6803     \bool_lazy_all:nTF
6804     {
6805         { \str_if_empty_p:N \l_@@_letter_str }
6806         { \str_if_empty_p:N \l_@@_command_str }
6807         { \str_if_empty_p:N \l_@@_cccommand_str }
6808     }
6809     { \@@_error:n { No~letter~and~no~command } }
6810     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6811 }
6812 \keys_define:nn { nicematrix / custom-line }
6813 {
6814     letter .str_set:N = \l_@@_letter_str ,
6815     letter .value_required:n = true ,
6816     command .str_set:N = \l_@@_command_str ,
6817     command .value_required:n = true ,
6818     ccommand .str_set:N = \l_@@_cccommand_str ,
6819     ccommand .value_required:n = true ,
6820 }
6821 \cs_new_protected:Npn \@@_custom_line_i:n #1
6822 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6823 \bool_set_false:N \l_@@_tikz_rule_bool
6824 \bool_set_false:N \l_@@_dotted_rule_bool
6825 \bool_set_false:N \l_@@_color_bool
6826 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6827 \bool_if:NT \l_@@_tikz_rule_bool
6828 {
6829   \IfPackageLoadedF { tikz }
6830   { \@@_error:n { tikz~in~custom-line~without~tikz } }
6831   \bool_if:NT \l_@@_color_bool
6832   { \@@_error:n { color~in~custom-line~with~tikz } }
6833 }
6834 \bool_if:NT \l_@@_dotted_rule_bool
6835 {
6836   \int_compare:nNtT { \l_@@_multiplicity_int } > { \c_one_int }
6837   { \@@_error:n { key~multiplicity~with~dotted } }
6838 }
6839 \str_if_empty:NF \l_@@_letter_str
6840 {
6841   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6842   { \@@_error:n { Several~letters } }
6843   {
6844     \tl_if_in:NoTF
6845     \c_@@_forbidden_letters_str
6846     \l_@@_letter_str
6847     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6848     {

```

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.

```

6849 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6850 { \@@_v_custom_line:nn { #1 } }
6851 }
6852 }
6853 }
6854 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6855 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6856 }
6857 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6858 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6859 \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.

```

6860 \keys_define:nn { nicematrix / custom-line-bis }
6861 {
6862   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6863   multiplicity .initial:n = 1 ,
6864   multiplicity .value_required:n = true ,
6865   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6866   color .value_required:n = true ,
6867   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6868   tikz .value_required:n = true ,
6869   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6870   dotted .value_forbidden:n = true ,
6871   total-width .code:n = { } ,
6872   total-width .value_required:n = true ,
6873   width .code:n = { } ,

```

```

6874 width .value_required:n = true ,
6875 sep-color .code:n = { } ,
6876 sep-color .value_required:n = true ,
6877 unknown .code:n =
6878   \@@_unknown_key:nn
6879   { nicematrix / custom-line-bis }
6880   { Unknown-key-for-custom-line }
6881 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6882 \bool_new:N \l_@@_dotted_rule_bool
6883 \bool_new:N \l_@@_tikz_rule_bool
6884 \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`.

```

6885 \keys_define:nn { nicematrix / custom-line-width }
6886 {
6887   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6888   multiplicity .initial:n = 1 ,
6889   multiplicity .value_required:n = true ,
6890   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6891   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6892   \bool_set_true:N \l_@@_total_width_bool ,
6893   total-width .value_required:n = true ,
6894   width .meta:n = { total-width = #1 } ,
6895   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6896 }

```

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`).

```

6897 \cs_new_protected:Npn \@@_h_custom_line:n #1
6898 {

```

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`).

```

6899   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6900   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6901 }

```

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`).

```

6902 \cs_new_protected:Npn \@@_c_custom_line:n #1
6903 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6904   \exp_args:Nc \NewExpandableDocumentCommand
6905   { nicematrix - \l_@@_ccommand_str }
6906   { 0 { } m }
6907   {
6908     \noalign
6909     {
6910       \@@_compute_rule_width:n { #1 , ##1 }
6911       \skip_vertical:n { \l_@@_rule_width_dim }
6912       \clist_map_inline:nn
6913       { ##2 }
6914       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }

```

```

6915     }
6916   }
6917   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6918 }

```

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*.

```

6919 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6920 {
6921   \tl_if_in:nnTF { #2 } { - }
6922   { \@@_cut_on_hyphen:w #2 \q_stop }
6923   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6924   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6925   {
6926     \@@_hline:n
6927     {
6928       #1 ,
6929       start = \l_tmpa_tl ,
6930       end = \l_tmpb_tl ,
6931       position = \int_eval:n { \c@iRow + 1 } ,
6932       total-width = \dim_use:N \l_@@_rule_width_dim
6933     }
6934   }
6935 }

6936 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6937 {
6938   \bool_set_false:N \l_@@_tikz_rule_bool
6939   \bool_set_false:N \l_@@_total_width_bool
6940   \bool_set_false:N \l_@@_dotted_rule_bool
6941   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6942   \bool_if:NF \l_@@_total_width_bool
6943   {
6944     \bool_if:NTF \l_@@_dotted_rule_bool
6945     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6946     {
6947       \bool_if:NF \l_@@_tikz_rule_bool
6948       {
6949         \dim_set:Nn \l_@@_rule_width_dim
6950         {
6951           \arrayrulewidth * \l_@@_multiplicity_int
6952           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6953         }
6954       }
6955     }
6956   }
6957 }

```

The following constructions aims to allow cumulative blocks of options between square brackets such as in `I[color=blue][tikz=dashed]`.

```

6958 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6959 {
6960   \str_if_eq:nnTF { #2 } { [ ] }
6961   { \@@_v_custom_line_i:nw { #1 } [ ] }
6962   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6963 }
6964 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6965 { \@@_v_custom_line:nn { #1 , #2 } }
6966 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6967 {
6968   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6969   \tl_gput_right:Ne \g_@@_array_preamble_tl

```

```

6970     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6971 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6972 {
6973     \@@_vline:n
6974     {
6975         #2 ,
6976         position = \int_eval:n { \c@jCol + 1 } ,
6977         total-width = \dim_use:N \l_@@_rule_width_dim
6978     }
6979 }
6980 \@@_rec_preamble:n #1
6981 }
6982 \@@_custom_line:n
6983 { 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.

```

6984 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6985 {
6986     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6987     {
6988         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6989         {
6990             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6991             {
6992                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6993                 { \bool_gset_false:N \g_tmpa_bool }
6994             }
6995         }
6996     }
6997 }

```

The same for vertical rules.

```

6998 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6999 {
7000     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
7001     {
7002         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
7003         {
7004             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
7005             {
7006                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7007                 { \bool_gset_false:N \g_tmpa_bool }
7008             }
7009         }
7010     }
7011 }

7012 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7013 {
7014     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
7015     {
7016         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
7017         {
7018             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
7019             { \bool_gset_false:N \g_tmpa_bool }
7020             {
7021                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
7022                 { \bool_gset_false:N \g_tmpa_bool }
7023             }

```



```

7024     }
7025   }
7026 }
7027 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7028 {
7029   \int_compare:nNtT { \l_tmpa_tl } > { #1 - 1 }
7030   {
7031     \int_compare:nNtT { \l_tmpa_tl } < { #3 + 1 }
7032     {
7033       \int_compare:nNtTF { \l_tmpb_tl } = { #2 }
7034       { \bool_gset_false:N \g_tmpa_bool }
7035       {
7036         \int_compare:nNtT { \l_tmpb_tl } = { #4 + 1 }
7037         { \bool_gset_false:N \g_tmpa_bool }
7038       }
7039     }
7040   }
7041 }

```

## 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.

```

7042 \cs_new_protected:Npn \@@_compute_corners:
7043 {
7044   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7045   { \@@_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`).

```

7046   \clist_clear:N \l_@@_corners_cells_clist
7047   \clist_map_inline:Nn \l_@@_corners_clist
7048   {
7049     \str_case:nnF { ##1 }
7050     {
7051       { NW }
7052       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7053       { NE }
7054       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7055       { SW }
7056       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7057       { SE }
7058       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7059     }
7060     { \@@_error:nn { bad~corner } { ##1 } }
7061   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7062   \clist_if_empty:NF \l_@@_corners_cells_clist
7063   {

```

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.

```

7064     \tl_gput_right:Ne \g_@@_aux_tl
7065     {
7066       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist

```

```

7067         { \l_@@_corners_cells_clist }
7068     }
7069 }
7070 }

7071 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7072 {
7073     \int_step_inline:nnn { #1 } { #3 }
7074     {
7075         \int_step_inline:nnn { #2 } { #4 }
7076         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7077     }
7078 }

7079 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7080 {
7081     \cs_if_exist:cTF
7082     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7083     { \prg_return_true: }
7084     { \prg_return_false: }
7085 }

```

“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.

```

7086 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7087 {

```

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.

```

7088     \bool_set_false:N \l_tmpa_bool
7089     \int_zero_new:N \l_@@_last_empty_row_int
7090     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7091     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7092     {
7093         \bool_lazy_or:nnTF
7094         {
7095             \cs_if_exist_p:c
7096             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7097         }
7098         { \@@_if_in_block_p:nn { ##1 } { #2 } }
7099         { \bool_set_true:N \l_tmpa_bool }
7100         {
7101             \bool_if:NF \l_tmpa_bool
7102             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7103         }
7104     }

```

Now, you determine the last empty cell in the row of number 1.

```

7105 \bool_set_false:N \l_tmpa_bool
7106 \int_zero_new:N \l_@@_last_empty_column_int
7107 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7108 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7109 {
7110   \bool_lazy_or:nnTF
7111   {
7112     \cs_if_exist_p:c
7113     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7114   }
7115   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7116   { \bool_set_true:N \l_tmpa_bool }
7117   {
7118     \bool_if:NF \l_tmpa_bool
7119     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7120   }
7121 }

```

Now, we loop over the rows.

```

7122 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
7123 {

```

We treat the row number ##1 with another loop.

```

7124 \bool_set_false:N \l_tmpa_bool
7125 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7126 {
7127   \bool_lazy_or:nnTF
7128   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7129   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7130   { \bool_set_true:N \l_tmpa_bool }
7131   {
7132     \bool_if:NF \l_tmpa_bool
7133     {
7134       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7135       \clist_put_right:Nn
7136       \l_@@_corners_cells_clist
7137       { ##1 - #####1 }
7138       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7139     }
7140   }
7141 }
7142 }
7143 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7144 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7145 \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.

```

7146 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

7147 \keys_define:nn { nicematrix / NiceMatrixBlock }
7148 {
7149   auto-columns-width .code:n =
7150   {
7151     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7152     \dim_gzero_new:N \g_@@_max_cell_width_dim
7153     \bool_set_true:N \l_@@_auto_columns_width_bool
7154   }
7155 }

7156 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7157 {
7158   \int_gincr:N \g_@@_NiceMatrixBlock_int
7159   \dim_zero:N \l_@@_columns_width_dim
7160   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7161   \bool_if:NT \l_@@_block_auto_columns_width_bool
7162   {
7163     \cs_if_exist:cT
7164     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7165     {
7166       \dim_set:Nn \l_@@_columns_width_dim
7167       {
7168         \use:c
7169         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7170       }
7171     }
7172   }
7173 }

```

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`).

```

7174 {
7175   \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.

```

7176   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7177   {
7178     \bool_if:NT \l_@@_block_auto_columns_width_bool
7179     {
7180       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7181       \iow_shipout:Ne \@mainaux
7182       {
7183         \cs_gset:cpn
7184         { @@ _ 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.

```

7185         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7186       }
7187       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7188     }
7189   }
7190   \ignorespacesafterend
7191 }

```

## 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).

```

7192 \cs_new_protected:Npn \@@_create_extra_nodes:
7193 {
7194   \bool_if:nTF \l_@@_medium_nodes_bool
7195   {
7196     \bool_if:NTF \l_@@_no_cell_nodes_bool
7197     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7198     {
7199       \bool_if:NTF \l_@@_large_nodes_bool
7200       \@@_create_medium_and_large_nodes:
7201       \@@_create_medium_nodes:
7202     }
7203   }
7204   {
7205     \bool_if:NT \l_@@_large_nodes_bool
7206     {
7207       \bool_if:NTF \l_@@_no_cell_nodes_bool
7208       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7209       \@@_create_large_nodes:
7210     }
7211   }
7212 }

```

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`.

```

7213 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7214 {
7215   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7216   {
7217     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7218     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7219     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7220     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7221   }
7222   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7223   {
7224     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7225     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7226     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7227     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7228   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7229 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7230 {
7231     \int_step_variable:nnNn
7232     \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.

```

7233 {
7234     \cs_if_exist:cT
7235     { 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`.

```

7236 {
7237     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7238     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7239     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7240     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7241     {
7242         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7243         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7244     }

```

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`.

```

7245     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7246     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7247     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7248     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7249     {
7250         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7251         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7252     }
7253 }
7254 }
7255 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7256 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7257 {
7258     \dim_compare:nNnT
7259     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7260     {
7261         \@@_qpoint:n { row - \@@_i: - base }
7262         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7263         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7264     }
7265 }
7266 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7267 {
7268     \dim_compare:nNnT
7269     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7270     {
7271         \@@_qpoint:n { col - \@@_j: }
7272         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7273         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7274     }
7275 }
7276 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7277 \cs_new_protected:Npn \@@_create_medium_nodes:
7278 {
7279   \pgfpicture
7280   \pgfrememberpicturepositiononpagetrue
7281   \pgf@relevantforpicturesizefalse
7282   \@@_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”.

```

7283   \tl_set:Nn \l_@@_suffix_tl { -medium }
7284   \@@_create_nodes:
7285   \endpgfpicture
7286 }

```

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:`.

```

7287 \cs_new_protected:Npn \@@_create_large_nodes:
7288 {
7289   \pgfpicture
7290   \pgfrememberpicturepositiononpagetrue
7291   \pgf@relevantforpicturesizefalse
7292   \@@_computations_for_medium_nodes:
7293   \@@_computations_for_large_nodes:
7294   \tl_set:Nn \l_@@_suffix_tl { - large }
7295   \@@_create_nodes:
7296   \endpgfpicture
7297 }
7298 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7299 {
7300   \pgfpicture
7301   \pgfrememberpicturepositiononpagetrue
7302   \pgf@relevantforpicturesizefalse
7303   \@@_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”.

```

7304   \tl_set:Nn \l_@@_suffix_tl { - medium }
7305   \@@_create_nodes:
7306   \@@_computations_for_large_nodes:
7307   \tl_set:Nn \l_@@_suffix_tl { - large }
7308   \@@_create_nodes:
7309   \endpgfpicture
7310 }

```

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.

```

7311 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7312 {
7313   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7314   \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`.

```

7315   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7316   {
7317     \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:`

```

7318     {
7319     (
7320         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7321         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7322     )
7323     / 2
7324     }
7325     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7326     { l_@@_row _ \@@_i: _ min _ dim }
7327 }
7328 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7329 {
7330     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7331     {
7332     (
7333         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7334         \dim_use:c
7335         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7336     )
7337     / 2
7338     }
7339     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7340     { l_@@_column _ \@@_j: _ max _ dim }
7341 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7342     \dim_sub:cn
7343     { l_@@_column _ 1 _ min _ dim }
7344     \l_@@_left_margin_dim
7345     \dim_add:cn
7346     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7347     \l_@@_right_margin_dim
7348 }

```

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).

```

7349 \cs_new_protected:Npn \@@_create_nodes:
7350 {
7351     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7352     {
7353         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7354         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7355         \@@_pgf_rect_node:nnnnn
7356         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7357         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7358         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7359         { \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } }
7360         { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7361         \str_if_empty:NF \l_@@_name_str
7362         {
7363             \pgfnodealias
7364             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7365             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7366         }
7367     }
7368 }
7369 \int_step_inline:nn { \c@iRow }

```



```

7370 {
7371   \pgfnodealias
7372   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7373   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7374 }
7375 \int_step_inline:nn { \c@jCol }
7376 {
7377   \pgfnodealias
7378   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7379   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7380 }
7381 \pgfnodealias % added 2025-04-05
7382 { \@@_env: - last - last \l_@@_suffix_tl }
7383 { \@@_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  $n$ .

```

7384   \seq_map_pairwise_function:NNN
7385   \g_@@_multicolumn_cells_seq
7386   \g_@@_multicolumn_sizes_seq
7387   \@@_node_for_multicolumn:nn
7388 }

7389 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7390 {
7391   \cs_set_nopar:Npn \@@_i: { #1 }
7392   \cs_set_nopar:Npn \@@_j: { #2 }
7393 }

```

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”).

```

7394 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7395 {
7396   \@@_extract_coords_values: #1 \q_stop
7397   \@@_pgf_rect_node:nnnnn
7398   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7399   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7400   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7401   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7402   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7403   \str_if_empty:NF \l_@@_name_str
7404   {
7405     \pgfnodealias
7406     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7407     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7408   }
7409 }

```

## 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).

```

7410 \keys_define:nn { nicematrix / Block / FirstPass }
7411 {
7412   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7413     \bool_set_true:N \l_@@_p_block_bool ,
7414   j .value_forbidden:n = true ,
7415   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7416   l .value_forbidden:n = true ,
7417   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7418   r .value_forbidden:n = true ,
7419   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7420   c .value_forbidden:n = true ,
7421   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7422   L .value_forbidden:n = true ,
7423   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,
7424   R .value_forbidden:n = true ,
7425   C .code:n = \str_set:Nn \l_@@_hpos_block_str C ,
7426   C .value_forbidden:n = true ,
7427   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7428   t .value_forbidden:n = true ,
7429   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7430   T .value_forbidden:n = true ,
7431   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7432   b .value_forbidden:n = true ,
7433   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7434   B .value_forbidden:n = true ,
7435   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7436   m .value_forbidden:n = true ,
7437   v-center .meta:n = m ,
7438   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7439   p .value_forbidden:n = true ,
7440   color .code:n =
7441     \@@_color:n { #1 }
7442     \tl_set_rescan:Nnn
7443       \l_@@_draw_tl
7444       { \char_set_catcode_other:N ! }
7445       { #1 } ,
7446   color .value_required:n = true ,
7447   respect-arraystretch .code:n =
7448     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7449   respect-arraystretch .value_forbidden:n = true ,
7450 }

```

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.

```

7451 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7452 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7453 {

```

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).

```

7454   \tl_if_blank:nTF { #2 }
7455   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7456   {
7457     \tl_if_in:nnTF { #2 } { - }
7458     {
7459       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7460       \@@_Block_i_czech:w \@@_Block_i:w
7461       #2 \q_stop
7462     }
7463     {
7464       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7465         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7466     }
7467 }
7468 { #1 } { #3 } { #4 }
7469 \ignorespaces
7470 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7471 \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`.

```

7472 {
7473   \char_set_catcode_active:N -
7474   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7475 }

```

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.

```

7476 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7477 {

```

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).

```

7478   \bool_lazy_or:nnTF
7479     { \tl_if_blank_p:n { #1 } }
7480     { \str_if_eq_p:ee { * } { #1 } }
7481     { \int_set:Nn \l_tmpa_int { 100 } }
7482     { \int_set:Nn \l_tmpa_int { #1 } }
7483   \bool_lazy_or:nnTF
7484     { \tl_if_blank_p:n { #2 } }
7485     { \str_if_eq_p:ee { * } { #2 } }
7486     { \int_set:Nn \l_tmpb_int { 100 } }
7487     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7488   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7489   {
7490     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7491     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7492     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7493   }
7494   { \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.

```

7495   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7496   \tl_set:Ne \l_tmpa_tl
7497   {
7498     { \int_use:N \c@iRow }
7499     { \int_use:N \c@jCol }
7500     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7501     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7502   }

```

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).

```

7503     \bool_set_false:N \l_tmpa_bool
7504     \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7505     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7506     \bool_case:nF
7507     {
7508         \l_tmpa_bool                                { \@@_Block_vii:eennn }
7509         \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.

```

7510         \l_@@_X_bool                                { \@@_Block_v:eennn }
7511         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7512         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7513         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7514     }
7515     { \@@_Block_v:eennn }
7516     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7517 }

```

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.

```

7518 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7519 {
7520     \int_gincr:N \g_@@_block_box_int
7521     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7522     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7523     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7524     {
7525         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7526         {
7527             \@@_actually_diagbox:nnnnnn
7528             { \int_use:N \c@iRow }
7529             { \int_use:N \c@jCol }
7530             { \int_eval:n { \c@iRow + #1 - 1 } }
7531             { \int_eval:n { \c@jCol + #2 - 1 } }
7532             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7533             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7534         }
7535     }
7536     \box_gclear_new:c
7537     { g_@@_block_box_int _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!

```

7538   \hbox_gset:cn
7539   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7540   {

```

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`).

```

7541   \tl_if_empty:NTF \l_@@_color_tl
7542   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7543   { \@@_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`.

```

7544   \int_compare:nNnT { #1 } = { \c_one_int }
7545   {
7546       \int_if_zero:nTF { \c@iRow }
7547       {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7548   \cs_set_eq:NN \Block \@@_NullBlock:
7549   \l_@@_code_for_first_row_tl
7550   }
7551   {
7552       \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7553       {
7554           \cs_set_eq:NN \Block \@@_NullBlock:
7555           \l_@@_code_for_last_row_tl
7556       }
7557   }
7558   \g_@@_row_style_tl
7559   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7560   \@@_reset_arraystretch:
7561   \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7562   #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.).

```
7563 \@@_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}`.

```
7564 \bool_if:NTF \l_@@_tabular_bool
7565 {
7566   \bool_lazy_all:nTF
7567   {
7568     { \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.

```
7569   {
7570     ! \dim_compare_p:nNn
7571     { \l_@@_col_width_dim } < { \c_zero_dim }
7572   }
7573   { ! \g_@@_rotate_bool }
7574 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7575 {
7576   \use:e
7577   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7578 \exp_not:N \begin { minipage }
7579 [ \str_lowercase:f \l_@@_vpos_block_str ]
7580 { \l_@@_col_width_dim }
7581 \str_case:on \l_@@_hpos_block_str
7582 { c \centering r \raggedleft l \raggedright }
7583 }
7584 #5
7585 \end { minipage }
7586 }
```

In the other cases, we use a `{tabular}`.

```
7587 {
7588   \use:e
7589   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7590 \exp_not:N \begin { tabular }
7591 [ \str_lowercase:f \l_@@_vpos_block_str ]
7592 { @ { } \l_@@_hpos_block_str @ { } }
7593 }
7594 #5
7595 \end { tabular }
7596 }
7597 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7598 {
7599   $ % $
7600   \use:e
7601   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7602         \exp_not:N \begin { array }
7603         [ \str_lowercase:f \l_@@_vpos_block_str ]
7604         { @ { } \l_@@_hpos_block_str @ { } }
7605     }
7606     #5
7607     \end { array }
7608     $ % $
7609 }
7610 }
```

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).

```

7611     \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.

```

7612     \int_compare:nNnT { #2 } = { \c_one_int }
7613     {
7614         \dim_gset:Nn \g_@@_blocks_wd_dim
7615         {
7616             \dim_max:nn
7617             { \g_@@_blocks_wd_dim }
7618             {
7619                 \box_wd:c
7620                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7621             }
7622         }
7623     }
```

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.

```

7624     \int_compare:nNnT { #1 } = { \c_one_int }
7625     {
7626         \bool_lazy_any:nT
7627         {
7628             { \str_if_empty_p:N \l_@@_vpos_block_str }
7629             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7630             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7631         }
7632         { \@@_adjust_blocks_ht_dp: }
7633     }
7634     \seq_gput_right:Ne \g_@@_blocks_seq
7635     {
7636         \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.

```

7637     {
7638         \exp_not:n { #3 } ,
7639         \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

7640     \bool_if:NT \g_@@_rotate_bool
7641     {
7642         \bool_if:NTF \g_@@_rotate_c_bool
7643         { m }
7644         {
```

```

7645         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7646         { T }
7647     }
7648 }
7649 }
7650 {
7651     \box_use_drop:c
7652     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7653 }
7654 }
7655 \bool_set_false:N \g_@@_rotate_c_bool
7656 }
7657 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7658 {
7659     \dim_gset:Nn \g_@@_blocks_ht_dim
7660     {
7661         \dim_max:nn
7662         { \g_@@_blocks_ht_dim }
7663         {
7664             \box_ht:c
7665             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7666         }
7667     }
7668     \dim_gset:Nn \g_@@_blocks_dp_dim
7669     {
7670         \dim_max:nn
7671         { \g_@@_blocks_dp_dim }
7672         {
7673             \box_dp:c
7674             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7675         }
7676     }
7677 }
7678 \cs_new:Npn \@@_adjust_hpos_rotate:
7679 {
7680     \bool_if:NT \g_@@_rotate_bool
7681     {
7682         \str_set:Ne \l_@@_hpos_block_str
7683         {
7684             \bool_if:NTF \g_@@_rotate_c_bool
7685             { c }
7686             {
7687                 \str_case:onF \l_@@_vpos_block_str
7688                 { b l B l t r T r }
7689                 {
7690                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7691                     { r }
7692                     { l }
7693                 }
7694             }
7695         }
7696     }
7697 }
7698 \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*.

```

7699 \cs_new_protected:Npn \@@_rotate_box_of_block:
7700 {
7701     \box_grotate:cn

```



```

7702     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7703     { 90 }
7704 \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
7705 {
7706     \vbox_gset_top:cn
7707     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7708     {
7709         \skip_vertical:n { 0.8 ex }
7710         \box_use:c
7711         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7712     }
7713 }
7714 \bool_if:NT \g_@@_rotate_c_bool
7715 {
7716     \hbox_gset:cn
7717     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7718     {
7719         $ % $
7720         \vcenter
7721         {
7722             \box_use:c
7723             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7724         }
7725         $ % $
7726     }
7727 }
7728 }

```

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.

```

7729 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7730 {
7731     \seq_gput_right:Ne \g_@@_blocks_seq
7732     {
7733         \l_tmpa_tl
7734         { \exp_not:n { #3 } }
7735         {
7736             \bool_if:NTF \l_@@_tabular_bool
7737             {
7738                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7739     \@@_reset_arraystretch:
7740     \exp_not:n
7741     {
7742         \dim_zero:N \extrarowheight
7743         #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).

```

7744     \tag_if_active:T { \tag_stop:n { table } }
7745     \use:e
7746     {
7747         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7748         { @ { } \l_@@_hpos_block_str @ { } }

```

```

7749         }
7750         #5
7751     \end { tabular }
7752 }
7753 \group_end:
7754 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7755 {
7756     \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7757     \@@_reset_arraystretch:
7758     \exp_not:n
7759     {
7760         \dim_zero:N \extrarowheight
7761         #4
7762         $ % $
7763         \use:e
7764         {
7765             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7766             { @ { } \l_@@_hpos_block_str @ { } }
7767         }
7768         #5
7769         \end { array }
7770         $ % $
7771     }
7772     \group_end:
7773 }
7774 }
7775 }
7776 }
7777 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

7778 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7779 {
7780     \seq_gput_right:Ne \g_@@_blocks_seq
7781     {
7782         \l_tmpa_tl
7783         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7784         { { \exp_not:n { #4 #5 } } }
7785     }
7786 }
7787 \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.

```

7788 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7789 {
7790     \seq_gput_right:Ne \g_@@_blocks_seq
7791     {
7792         \l_tmpa_tl
7793         { \exp_not:n { #3 } }
7794         { \exp_not:n { #4 #5 } }
7795     }
7796 }
7797 \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).

```

7798 \keys_define:n { nicematrix / Block / SecondPass }
7799 {
7800     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7801     ampersand-in-blocks .default:n = true ,
7802     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7803     tikz .code:n =
7804         \IfPackageLoadedTF { tikz }
7805         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7806         { \@@_error:n { tikz~key~without~tikz } } ,
7807     tikz .value_required:n = true ,
7808     fill .code:n =
7809         \tl_set_rescan:Nnn
7810         \l_@@_fill_tl
7811         { \char_set_catcode_other:N ! }
7812         { #1 } ,
7813     fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

7814     opacity .tl_set:N = \l_@@_opacity_tl ,
7815     opacity .value_required:n = true ,
7816     draw .code:n =
7817         \tl_set_rescan:Nnn
7818         \l_@@_draw_tl
7819         { \char_set_catcode_other:N ! }
7820         { #1 } ,
7821     draw .default:n = default ,
7822     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7823     rounded-corners .default:n = 4 pt ,
7824     color .code:n =
7825         \@@_color:n { #1 }
7826         \tl_set_rescan:Nnn
7827         \l_@@_draw_tl
7828         { \char_set_catcode_other:N ! }
7829         { #1 } ,
7830     borders .clist_set:N = \l_@@_borders_clist ,
7831     borders .value_required:n = true ,
7832     hvlines .meta:n = { vlines , hlines } ,
7833     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7834     vlines .default:n = true ,
7835     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7836     hlines .default:n = true ,
7837     line-width .dim_set:N = \l_@@_line_width_dim ,
7838     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7839     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7840         \bool_set_true:N \l_@@_p_block_bool ,
7841     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7842     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7843     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7844     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7845         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7846     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7847         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7848     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7849         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7850     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7851     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7852     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7853     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7854     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7855     m .value_forbidden:n = true ,
7856     v-center .meta:n = m ,

```

```

7857 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7858 p .value_forbidden:n = true ,
7859 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7860 name .value_required:n = true ,
7861 name .initial:n = ,
7862 respect-arraystretch .code:n =
7863   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7864 respect-arraystretch .value_forbidden:n = true ,
7865 transparent .bool_set:N = \l_@@_transparent_bool ,
7866 transparent .default:n = true ,
7867 transparent .initial:n = false ,
7868 unknown .code:n =
7869   \@@_unknown_key:nn
7870     { nicematrix / Block / SecondPass }
7871     { Unknown-key-for-Block }
7872 }

```

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.

```

7873 \cs_new_protected:Npn \@@_draw_blocks:
7874 {
7875   \bool_if:NTF \c_@@_revtex_bool
7876     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7877     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7878   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7879 }
7880 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7881 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7882   \int_zero:N \l_@@_last_row_int
7883   \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”).

```

7884   \int_compare:nNnTF { #3 } > { 98 }
7885     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7886     { \int_set:Nn \l_@@_last_row_int { #3 } }
7887   \int_compare:nNnTF { #4 } > { 98 }
7888     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7889     { \int_set:Nn \l_@@_last_col_int { #4 } }
7890   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7891     {
7892       \bool_lazy_and:nnTF
7893         { \l_@@_preamble_bool }
7894         {
7895           \int_compare_p:n
7896             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7897         }
7898     {
7899       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7900       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7901       \@@_msg_redirect_name:nn { columns-not-used } { none }
7902     }
7903     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7904 }

```

```

7905 {
7906   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7907   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7908   {
7909     \@@_Block_v:nneenn
7910     { #1 }
7911     { #2 }
7912     { \int_use:N \l_@@_last_row_int }
7913     { \int_use:N \l_@@_last_col_int }
7914     { #5 }
7915     { #6 }
7916   }
7917 }
7918 }

```

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.

```

7919 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7920 {

```

The group is for the keys.

```

7921   \group_begin:
7922   \int_compare:nNnT { #1 } = { #3 }
7923   { \str_set:Nn \l_@@_vpos_block_str { t } }
7924   \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`.

```

7925   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7926   \bool_lazy_and:nnT
7927   { \l_@@_vlines_block_bool }
7928   { ! \l_@@_ampersand_bool }
7929   {
7930     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7931     {
7932       \@@_vlines_block:nnn
7933       { \exp_not:n { #5 } }
7934       { #1 - #2 }
7935       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7936     }
7937   }
7938   \bool_if:NT \l_@@_hlines_block_bool
7939   {
7940     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7941     {
7942       \@@_hlines_block:nnn
7943       { \exp_not:n { #5 } }
7944       { #1 - #2 }
7945       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7946     }
7947   }
7948   \bool_if:NF \l_@@_transparent_bool
7949   {
7950     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7951     {

```

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).

```

7952     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7953     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7954   }
7955 }

```

```

7956 \tl_if_empty:NF \l_@@_draw_tl
7957 {
7958   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7959   { \@@_error:n { hlines~with~color } }
7960   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7961   {
7962     \@@_stroke_block:nnn
7963     { \exp_not:n { #5 } }
7964     { #1 - #2 }
7965     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7966   }
7967   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7968   { { #1 } { #2 } { #3 } { #4 } }
7969 }
7970 \clist_if_empty:NF \l_@@_borders_clist
7971 {
7972   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7973   {
7974     \@@_stroke_borders_block:nnn
7975     { \exp_not:n { #5 } }
7976     { #1 - #2 }
7977     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7978   }
7979 }
7980 \tl_if_empty:NF \l_@@_fill_tl
7981 {
7982   \@@_add_opacity_to_fill:
7983   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7984   {
7985     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7986     { #1 - #2 }
7987     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7988     { \dim_use:N \l_@@_rounded_corners_dim }
7989   }
7990 }
7991 \seq_if_empty:NF \l_@@_tikz_seq
7992 {
7993   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7994   {
7995     \@@_block_tikz:nnnnn
7996     { \seq_use:Nn \l_@@_tikz_seq { , } }
7997     { #1 }
7998     { #2 }
7999     { \int_use:N \l_@@_last_row_int }
8000     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of TikZ keys.

```

8001   }
8002 }
8003 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8004 {
8005   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8006   {
8007     \@@_actually_diagbox:nnnnnn
8008     { #1 }
8009     { #2 }
8010     { \int_use:N \l_@@_last_row_int }
8011     { \int_use:N \l_@@_last_col_int }
8012     { \exp_not:n { ##1 } }
8013     { \exp_not:n { ##2 } }

```

```

8014     }
8015 }

```

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.

```

8016 \pgfpicture
8017 \pgfrememberpicturepositiononpagetrue
8018 \pgf@relevantforpicturesizefalse
8019 \@@_qpoint:n { row - #1 }
8020 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8021 \@@_qpoint:n { col - #2 }
8022 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8023 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8024 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8025 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8026 \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.

```

8027 \@@_pgf_rect_node:nnnnn
8028 { \@@_env: - #1 - #2 - block }
8029 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8030 \str_if_empty:NF \l_@@_block_name_str
8031 {
8032   \pgfnodealias
8033   { \@@_env: - \l_@@_block_name_str }
8034   { \@@_env: - #1 - #2 - block }
8035   \str_if_empty:NF \l_@@_name_str
8036   {
8037     \pgfnodealias
8038     { \l_@@_name_str - \l_@@_block_name_str }
8039     { \@@_env: - #1 - #2 - block }
8040   }
8041 }

```

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.

```

8042 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8043 {
8044   \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.

```

8045     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8046     {

```

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.

```

8047         \cs_if_exist:cT
8048         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8049         {
8050             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8051             {
8052                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8053                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8054             }
8055         }
8056     }

```

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.

```

8057     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
8058     {
8059         \@@_qpoint:n { col - #2 }
8060         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8061     }
8062     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8063     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8064     {
8065         \cs_if_exist:cT
8066         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8067         {
8068             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8069             {
8070                 \pgfpointanchor
8071                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8072                 { east }
8073                 \dim_set:Nn \l_@@_tmpd_dim
8074                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
8075             }
8076         }
8077     }
8078     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
8079     {
8080         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8081         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8082     }
8083     \@@_pgf_rect_node:nnnn
8084     { \@@_env: - #1 - #2 - block - short }
8085     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8086 }

```

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.

```

8087     \bool_if:NT \l_@@_medium_nodes_bool
8088     {
8089         \@@_pgf_rect_node:nnn
8090         { \@@_env: - #1 - #2 - block - medium }
8091         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8092         {
8093             \pgfpointanchor
8094             { \@@_env:
8095                 - \int_use:N \l_@@_last_row_int
8096                 - \int_use:N \l_@@_last_col_int - medium
8097             }

```



```

8098         { south~east }
8099     }
8100 }
8101 \endpgfpicture
8102

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8103 \bool_if:NTF \l_@@_ampersand_bool
8104 {
8105     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8106     \int_zero_new:N \l_@@_split_int
8107     \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.

```

8108     \int_zero_new:N \l_@@_first_row_int
8109     \int_zero_new:N \l_@@_first_col_int
8110     \int_zero_new:N \l_@@_last_row_int
8111     \int_zero_new:N \l_@@_last_col_int
8112     \int_set:Nn \l_@@_first_row_int { #1 }
8113     \int_set:Nn \l_@@_first_col_int { #2 }
8114     \int_set:Nn \l_@@_last_row_int { #3 }
8115     \int_set:Nn \l_@@_last_col_int { #4 }

8116     \pgfpicture
8117     \pgfrememberpicturerepositiononpagetrue
8118     \pgf@relevantforpicturesizefalse
8119     \@@_qpoint:n { row - #1 }
8120     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8121     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8122     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8123     \@@_qpoint:n { col - #2 }
8124     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8125     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8126     \dim_set:Nn \l_tmpb_dim
8127     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8128     \bool_lazy_or:nnT
8129     { \l_@@_vlines_block_bool }
8130     { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8131     {
8132         \int_step_inline:nn { \l_@@_split_int - 1 }
8133         {
8134             \pgfpathmoveto
8135             {
8136                 \pgfpoint
8137                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8138                 \l_@@_tmpc_dim
8139             }
8140             \pgfpathlineto
8141             {
8142                 \pgfpoint
8143                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8144                 \l_@@_tmpd_dim
8145             }
8146             \CT@arc@
8147             \pgfsetlinewidth { 1.1 \arrayrulewidth }
8148             \pgfsetrectcap
8149             \pgfusepathqstroke
8150         }
8151     }
8152     \cs_set_eq:NN \cellcolor \@@_subcellcolor
8153     \int_zero_new:N \l_@@_split_i_int
8154     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }

```

```

8155 {
8156   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8157   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8158 }
8159 {
8160   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8161   {
8162     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8163     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8164   }
8165   {
8166     \bool_lazy_or:nnTF
8167     { \int_compare_p:nNn { #1 } = { #3 } }
8168     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8169     {
8170       \@@_qpoint:n { row - #1 - base }
8171       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8172     }
8173     {
8174       \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8175       {
8176         \@@_qpoint:n { row - #3 - base }
8177         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8178       }
8179       {
8180         \@@_qpoint:n { #1 - #2 - block }
8181         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8182       }
8183     }
8184   }
8185 }
8186 \int_step_inline:nn { \l_@@_split_int }
8187 {
8188   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8189   \int_set:Nn \l_@@_split_i_int { ##1 }
8190   \dim_set:Nn \col@sep
8191   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8192   \pgftransformshift
8193   {
8194     \pgfpoint
8195     {
8196       \l_tmpa_dim + ##1 \l_tmpb_dim -
8197       \str_case:on \l_@@_hpos_block_str
8198       {
8199         l { \l_tmpb_dim + \col@sep }
8200         c { 0.5 \l_tmpb_dim }
8201         r { \col@sep }
8202       }
8203     }
8204     { \l_@@_tmpc_dim }
8205   }
8206   \pgfset { inner~sep = \c_zero_dim }
8207   \pgfnode
8208   { rectangle }
8209   {
8210     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8211     {
8212       \str_case:on \l_@@_hpos_block_str
8213       {
8214         l { north-west }
8215         c { north }
8216         r { north-east }

```

```

8217     }
8218   }
8219   {
8220     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8221     {
8222       \str_case:on \l_@@_hpos_block_str
8223       {
8224         l { south~west }
8225         c { south }
8226         r { south~east }
8227       }
8228     }
8229     {
8230       \bool_lazy_any:nTF
8231       {
8232         { \int_compare_p:nNn { #1 } = { #3 } }
8233         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8234         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8235       }
8236       {
8237         \str_case:on \l_@@_hpos_block_str
8238         {
8239           l { base~west }
8240           c { base }
8241           r { base~east }
8242         }
8243       }
8244       {
8245         \str_case:on \l_@@_hpos_block_str
8246         {
8247           l { west }
8248           c { center }
8249           r { east }
8250         }
8251       }
8252     }
8253   }
8254 }
8255 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8256 \group_end:
8257 }
8258 \endpgfpicture
8259 }

```

Now the case where there is no ampersand & in the content of the block.

```

8260 {
8261   \bool_if:NTF \l_@@_p_block_bool
8262   {

```

When the final user has used the key p, we have to compute the width.

```

8263   \pgfpicture
8264   \pgfrememberpicturepositiononpagetrue
8265   \pgf@relevantforpicturesizefalse
8266   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8267   {
8268     \@@_qpoint:n { col - #2 }
8269     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8270     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8271   }
8272   {
8273     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8274     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8275     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8276   }

```

```

8277         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8278     \endpgfpicture
8279     \hbox_set:Nn \l_@@_cell_box
8280     {
8281         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8282         { \g_tmpb_dim }
8283         \str_case:on \l_@@_hpos_block_str
8284         { c \centering r \raggedleft l \raggedright j { } }
8285         #6
8286         \end { minipage }
8287     }
8288 }
8289 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8290 \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.

```

8291 \pgfpicture
8292 \pgfrememberpicturepositiononpagetrue
8293 \pgf@relevantforpicturesizefalse
8294 \bool_lazy_any:nTF
8295 {
8296     { \str_if_empty_p:N \l_@@_vpos_block_str }
8297     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8298     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8299     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8300 }
8301 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8302     \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`.

```

8303     \bool_if:nT \g_@@_last_col_found_bool
8304     {
8305         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8306         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8307     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8308     \tl_set:Ne \l_tmpa_tl
8309     {
8310         \str_case:on \l_@@_vpos_block_str
8311         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8312         { } {
8313             \str_case:on \l_@@_hpos_block_str
8314             {
8315                 c { center }
8316                 l { west }
8317                 r { east }
8318                 j { center }
8319             }
8320         }
8321         c {
8322             \str_case:on \l_@@_hpos_block_str
8323             {
8324                 c { center }
8325                 l { west }
8326                 r { east }
8327                 j { center }

```

```

8328         }
8329
8330     }
8331     T {
8332         \str_case:on \l_@@_hpos_block_str
8333         {
8334             c { north }
8335             l { north~west }
8336             r { north~east }
8337             j { north }
8338         }
8339
8340     }
8341     B {
8342         \str_case:on \l_@@_hpos_block_str
8343         {
8344             c { south }
8345             l { south~west }
8346             r { south~east }
8347             j { south }
8348         }
8349
8350     }
8351 }
8352 }
8353 \pgftransformshift
8354 {
8355     \pgfpointanchor
8356     {
8357         \@@_env: - #1 - #2 - block
8358         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8359     }
8360     { \l_tmpa_tl }
8361 }
8362 \pgfset { inner~sep = \c_zero_dim }
8363 \pgfnode
8364 { rectangle }
8365 { \l_tmpa_tl }
8366 { \box_use_drop:N \l_@@_cell_box } { } { }
8367 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8368 {
8369     \pgfextracty \l_tmpa_dim
8370     {
8371         \@@_qpoint:n
8372         {
8373             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8374             - base
8375         }
8376     }
8377     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

8378 \pgfpointanchor
8379 {
8380     \@@_env: - #1 - #2 - block
8381     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8382 }
8383 {
8384     \str_case:on \l_@@_hpos_block_str
8385     {
8386         c { center }

```

```

8387         l { west }
8388         r { east }
8389         j { center }
8390     }
8391 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8392     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8393     \pgfset { inner~sep = \c_zero_dim }
8394     \pgfnode
8395     { rectangle }
8396     {
8397         \str_case:on \l_@@_hpos_block_str
8398         {
8399             c { base }
8400             l { base~west }
8401             r { base~east }
8402             j { base }
8403         }
8404     }
8405     { \box_use_drop:N \l_@@_cell_box } { } { }
8406 }
8407 \endpgfpicture
8408 }
8409 \group_end:
8410 }
8411 \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).

```

8412 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8413 {
8414     \tl_if_empty:NF \l_@@_opacity_tl
8415     {
8416         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8417             {
8418                 \tl_set:Nc \l_@@_fill_tl
8419                 {
8420                     [ opacity = \l_@@_opacity_tl ,
8421                     \tl_tail:o \l_@@_fill_tl
8422                 }
8423             }
8424             {
8425                 \tl_set:Nc \l_@@_fill_tl
8426                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8427             }
8428         }
8429     }

```

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).

```

8430 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8431 {
8432     \group_begin:
8433     \tl_clear:N \l_@@_draw_tl
8434     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8435     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8436     \pgfpicture
8437     \pgfrememberpicturepositiononpagetrue
8438     \pgf@relevantforpicturesizefalse

```

```

8439 \tl_if_empty:NF \l_@@_draw_tl
8440 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8441 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8442 { \CT@arc@ }
8443 { \@@_color:o \l_@@_draw_tl }
8444 }
8445 \pgfsetcornersarced
8446 {
8447 \pgfpoint
8448 { \l_@@_rounded_corners_dim }
8449 { \l_@@_rounded_corners_dim }
8450 }
8451 \@@_cut_on_hyphen:w #2 \q_stop
8452 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8453 {
8454 \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8455 {
8456 \@@_qpoint:n { row - \l_tmpa_tl }
8457 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8458 \@@_qpoint:n { col - \l_tmpb_tl }
8459 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8460 \@@_cut_on_hyphen:w #3 \q_stop
8461 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8462 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8463 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8464 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8465 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8466 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8467 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8468 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8469 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8470 \pgfpathrectanglecorners
8471 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8472 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8473 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8474 { \pgfusepathqstroke }
8475 { \pgfusepath { stroke } }
8476 }
8477 }
8478 \endpgfpicture
8479 \group_end:
8480 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8481 \keys_define:nn { nicematrix / BlockStroke }
8482 {
8483 color .tl_set:N = \l_@@_draw_tl ,
8484 draw .code:n =
8485 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8486 draw .default:n = default ,
8487 line-width .dim_set:N = \l_@@_line_width_dim ,
8488 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8489 rounded-corners .default:n = 4 pt
8490 }

```

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).

```

8491 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8492 {

```

```

8493 \group_begin:
8494 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8495 \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8496 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8497 \@@_cut_on_hyphen:w #2 \q_stop
8498 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8499 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8500 \@@_cut_on_hyphen:w #3 \q_stop
8501 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8502 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8503 \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8504 {
8505   \use:e
8506   {
8507     \@@_vline:n
8508     {
8509       position = ##1 ,
8510       start = \l_@@_tmpc_tl ,
8511       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8512       total-width = \dim_use:N \l_@@_line_width_dim
8513     }
8514   }
8515 }
8516 \group_end:
8517 }

8518 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8519 {
8520   \group_begin:
8521   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8522   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8523   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8524   \@@_cut_on_hyphen:w #2 \q_stop
8525   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8526   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8527   \@@_cut_on_hyphen:w #3 \q_stop
8528   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8529   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8530   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8531   {
8532     \use:e
8533     {
8534       \@@_hline:n
8535       {
8536         position = ##1 ,
8537         start = \l_@@_tmpd_tl ,
8538         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8539         total-width = \dim_use:N \l_@@_line_width_dim
8540       }
8541     }
8542   }
8543   \group_end:
8544 }

```

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).

```

8545 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8546 {
8547   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8548   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8549   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8550   { \@@_error:n { borders~forbidden } }

```



```

8551 {
8552   \tl_clear_new:N \l_@@_borders_tikz_tl
8553   \keys_set:no
8554     { nicematrix / OnlyForTikzInBorders }
8555   \l_@@_borders_clist
8556   \@@_cut_on_hyphen:w #2 \q_stop
8557   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8558   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8559   \@@_cut_on_hyphen:w #3 \q_stop
8560   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8561   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8562   \@@_stroke_borders_block_i:
8563 }
8564 }

8565 \hook_gput_code:nnn { begindocument } { . }
8566 {
8567   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8568   {
8569     \c_@@_pgfortikzpicture_tl
8570     \@@_stroke_borders_block_ii:
8571     \c_@@_endpgfortikzpicture_tl
8572   }
8573 }

8574 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8575 {
8576   \pgfrememberpicturepositiononpagetrue
8577   \pgf@relevantforpicturesizefalse
8578   \CT@arc@
8579   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8580   \clist_if_in:NnT \l_@@_borders_clist { right }
8581     { \@@_stroke_vertical:n \l_tmpb_tl }
8582   \clist_if_in:NnT \l_@@_borders_clist { left }
8583     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8584   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8585     { \@@_stroke_horizontal:n \l_tmpa_tl }
8586   \clist_if_in:NnT \l_@@_borders_clist { top }
8587     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8588 }

8589 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8590 {
8591   tikz .code:n =
8592     \cs_if_exist:NTF \tikzpicture
8593     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8594     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8595   tikz .value_required:n = true ,
8596   top .code:n = ,
8597   bottom .code:n = ,
8598   left .code:n = ,
8599   right .code:n = ,
8600   unknown .code:n = \@@_error:n { bad~border }
8601 }

```

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).

```

8602 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8603 {
8604   \@@_qpoint:n \l_@@_tmpc_tl
8605   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8606   \@@_qpoint:n \l_tmpa_tl
8607   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8608   \@@_qpoint:n { #1 }
8609   \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

8610 {
8611   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8612   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8613   \pgfusepathqstroke
8614 }
8615 {
8616   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8617   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8618 }
8619 }

```

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).

```

8620 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8621 {
8622   \@@_qpoint:n \l_@@_tmpd_tl
8623   \clist_if_in:NnTF \l_@@_borders_clist { left }
8624   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8625   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8626   \@@_qpoint:n \l_tmpb_tl
8627   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8628   \@@_qpoint:n { #1 }
8629   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8630   {
8631     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8632     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8633     \pgfusepathqstroke
8634   }
8635   {
8636     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8637     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8638   }
8639 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

8640 \keys_define:nn { nicematrix / BlockBorders }
8641 {
8642   borders .clist_set:N = \l_@@_borders_clist ,
8643   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8644   rounded-corners .default:n = 4 pt ,
8645   line-width .dim_set:N = \l_@@_line_width_dim
8646 }

```

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.

```

8647 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8648 {
8649   \begin { tikzpicture }
8650   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8651   \clist_map_inline:nn { #1 }
8652   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8653 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8654 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8655 (
8656 [
8657 xshift = \dim_use:N \l_@@_offset_dim ,
8658 yshift = - \dim_use:N \l_@@_offset_dim
8659 ]
8660 #2 -| #3
8661 )
8662 rectangle
8663 (
8664 [
8665 xshift = - \dim_use:N \l_@@_offset_dim ,
8666 yshift = \dim_use:N \l_@@_offset_dim
8667 ]
8668 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8669 ) ;
8670 }
8671 \end { tikzpicture }
8672 }
8673 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8674 \keys_define:nn { nicematrix / SpecialOffset }
8675 { 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.

```

8676 \cs_new_protected:Npn \@@_NullBlock:
8677 { \@@_collect_options:n { \@@_NullBlock_i: } }
8678 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8679 { }

```

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.

```

8680 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8681 {
8682 \tl_gput_right:Ne \g_@@_pre_code_before_tl
8683 {

```

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`).

```

8684 \@@_subcellcolor:nnnnnnn
8685 {
8686 \tl_if_blank:nTF { #1 }
8687 { { \exp_not:n { #2 } } }
8688 { [ #1 ] { \exp_not:n { #2 } } }
8689 }
8690 { \int_use:N \l_@@_first_row_int } % first row of the block
8691 { \int_use:N \l_@@_first_col_int } % first column of the block
8692 { \int_use:N \l_@@_last_row_int } % last row of the block
8693 { \int_use:N \l_@@_last_col_int } % last column of the block
8694 { \int_use:N \l_@@_split_int }
8695 { \int_use:N \l_@@_split_i_int }
8696 }
8697 \ignorespaces
8698 }

8699 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8700 {
8701 \@@_color_opacity: #1

```

```

8702 \pgfpicture
8703 \pgf@relevantforpicturesizefalse
8704 \@@_qpoint:n { col - #3 }
8705 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8706 \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8707 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8708 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8709 \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8710 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8711 \@@_qpoint:n { row - #2 }
8712 \pgfpathrectanglecorners
8713   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8714   { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8715 \pgfusepathqfill
8716 \endpgfpicture
8717 }

```

## 27 How to draw the dotted lines transparently

```

8718 \cs_set_protected:Npn \@@_renew_matrix:
8719 {
8720   \RenewDocumentEnvironment { pmatrix } { } {
8721     { \pNiceMatrix }
8722     { \endpNiceMatrix }
8723   }
8724   \RenewDocumentEnvironment { vmatrix } { } {
8725     { \vNiceMatrix }
8726     { \endvNiceMatrix }
8727   }
8728   \RenewDocumentEnvironment { Vmatrix } { } {
8729     { \VNiceMatrix }
8730     { \endVNiceMatrix }
8731   }
8732   \RenewDocumentEnvironment { bmatrix } { } {
8733     { \bNiceMatrix }
8734     { \endbNiceMatrix }
8735   }
8736   \RenewDocumentEnvironment { Bmatrix } { } {
8737     { \BNiceMatrix }
8738     { \endBNiceMatrix }
8739   }
8740 }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8736 \keys_define:nn { nicematrix / Auto }
8737 {
8738   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8739   columns-type .value_required:n = true ,
8740   l .meta:n = { columns-type = l } ,
8741   r .meta:n = { columns-type = r } ,
8742   c .meta:n = { columns-type = c } ,
8743   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8744   delimiters / color .value_required:n = true ,
8745   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8746   delimiters / max-width .default:n = true ,
8747   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8748   delimiters .value_required:n = true ,
8749   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8750   rounded-corners .default:n = 4 pt
8751 }

```

```

8752 \NewDocumentCommand \AutoNiceMatrixWithDelims
8753 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8754 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8755 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8756 {

```

The group is for the protection of the keys.

```

8757 \group_begin:
8758 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8759 \use:e
8760 {
8761 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8762 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8763 [ \exp_not:o \l_tmpa_tl ]
8764 }
8765 \int_if_zero:nT { \l_@@_first_row_int }
8766 {
8767 \int_if_zero:nT { \l_@@_first_col_int } { & }
8768 \prg_replicate:nn { #4 - 1 } { & }
8769 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8770 }
8771 \prg_replicate:nn { #3 }
8772 {
8773 \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).

```

8774 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8775 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8776 }
8777 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8778 {
8779 \int_if_zero:nT { \l_@@_first_col_int } { & }
8780 \prg_replicate:nn { #4 - 1 } { & }
8781 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8782 }
8783 \end { NiceArrayWithDelims }
8784 \group_end:
8785 }

8786 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8787 {
8788 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8789 {
8790 \bool_gset_true:N \g_@@_delims_bool
8791 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8792 \AutoNiceMatrixWithDelims { #2 } { #3 }
8793 }
8794 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8795 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8796 {
8797 \group_begin:
8798 \bool_gset_false:N \g_@@_delims_bool
8799 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8800 \group_end:
8801 }

```

## 29 The redefinition of the command `\dotfill`

```

8802 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8803 \cs_new_protected:Npn \@@_dotfill:
8804 {

```

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}`).

```

8805 \@@_old_dotfill:
8806 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8807 }

```

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`.

```

8808 \cs_new_protected:Npn \@@_dotfill_i:
8809 {
8810 \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8811 { \@@_old_dotfill: }
8812 }

```

## 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.

```

8813 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8814 {
8815 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8816 {
8817 \@@_actually_diagbox:nnnnnn
8818 { \int_use:N \c@iRow }
8819 { \int_use:N \c@jCol }
8820 { \int_use:N \c@iRow }
8821 { \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.

```

8822 { \g_@@_row_style_tl \exp_not:n { #1 } }
8823 { \g_@@_row_style_tl \exp_not:n { #2 } }
8824 }

```

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`.

```

8825 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8826 {
8827 { \int_use:N \c@iRow }
8828 { \int_use:N \c@jCol }
8829 { \int_use:N \c@iRow }
8830 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8831 { }
8832 }
8833 }

```

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.

```

8834 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8835 {
8836   \pgfpicture
8837   \pgf@relevantforpicturesizefalse
8838   \pgfrememberpicturepositiononpagetrue
8839   \@@_qpoint:n { row - #1 }
8840   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8841   \@@_qpoint:n { col - #2 }
8842   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8843   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8844   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8845   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8846   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8847   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8848   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8849   {

```

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.

```

8850   \CT@arc@
8851   \pgfsetroundcap
8852   \pgfusepathqstroke
8853 }
8854 \pgfset { inner~sep = 1 pt }
8855 \pgfscope
8856 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8857 \pgfnode { rectangle } { south~west }
8858 {
8859   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8860   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8861   \end { minipage }
8862 }
8863 { }
8864 { }
8865 \endpgfscope
8866 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8867 \pgfnode { rectangle } { north~east }
8868 {
8869   \begin { minipage } { 20 cm }
8870   \raggedleft
8871   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8872   \end { minipage }
8873 }
8874 { }
8875 { }
8876 \endpgfpicture
8877 }

```

## 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. 89.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8878 \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 `\`.

```
8879 \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`.

```
8880 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8881 {
8882   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8883   \@@_CodeAfter_iv:n
8884 }
```

We catch the argument of the command `\end` (in `#1`).

```
8885 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8886 {
```

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.

```
8887   \str_if_eq:eeTF { \@currenvir } { #1 }
8888   { \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`.

```
8889   {
8890     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8891     \@@_CodeAfter_ii:n
8892   }
8893 }
```

## 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.

```
8894 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8895 {
8896   \pgfpicture
8897   \pgfrememberpicturepositiononpagetrue
8898   \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.

```
8899   \@@_qpoint:n { row - 1 }
8900   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8901   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8902   \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).

```

8903   \bool_if:nTF { #3 }
8904     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8905     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8906   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8907     {
8908       \cs_if_exist:cT
8909         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8910         {
8911           \pgfpointanchor
8912             { \@@_env: - ##1 - #2 }
8913             { \bool_if:nTF { #3 } { west } { east } }
8914           \dim_set:Nn \l_tmpa_dim
8915             {
8916               \bool_if:nTF { #3 }
8917                 { \dim_min:nn }
8918                 { \dim_max:nn }
8919               \l_tmpa_dim
8920               { \pgf@x }
8921             }
8922         }
8923     }

```

Now we can put the delimiter with a node of PGF.

```

8924   \pgfset { inner~sep = \c_zero_dim }
8925   \dim_zero:N \nulldelimiterspace
8926   \pgftransformshift
8927     {
8928       \pgfpoint
8929         { \l_tmpa_dim }
8930         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8931     }
8932   \pgfnode
8933     { rectangle }
8934     { \bool_if:nTF { #3 } { east } { west } }
8935     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8936       \nullfont
8937       $ % $
8938       \@@_color:o \l_@@_delimiters_color_tl
8939       \bool_if:nTF { #3 } { \left #1 } { \left . }
8940       \vcenter
8941         {
8942           \nullfont
8943           \hrule \@@height
8944             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8945             \@@depth \c_zero_dim
8946             \@@width \c_zero_dim
8947         }
8948       \bool_if:nTF { #3 } { \right . } { \right #1 }
8949       $ % $
8950     }
8951   { }
8952   { }
8953   \endpgfpicture
8954 }

```

### 33 The command `\SubMatrix`

```

8955 \keys_define:nn { nicematrix / sub-matrix }
8956 {
8957   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8958   extra-height .value_required:n = true ,
8959   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8960   left-xshift .value_required:n = true ,
8961   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8962   right-xshift .value_required:n = true ,
8963   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8964   xshift .value_required:n = true ,
8965   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8966   delimiters / color .value_required:n = true ,
8967   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8968   slim .default:n = true ,
8969   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8970   hlines .default:n = all ,
8971   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8972   vlines .default:n = all ,
8973   hvlines .meta:n = { hlines, vlines } ,
8974   hvlines .value_forbidden:n = true
8975 }
8976 \keys_define:nn { nicematrix }
8977 {
8978   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8979   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8980   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8981   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8982 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8983 \keys_define:nn { nicematrix / SubMatrix }
8984 {
8985   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8986   delimiters / color .value_required:n = true ,
8987   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8988   hlines .default:n = all ,
8989   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8990   vlines .default:n = all ,
8991   hvlines .meta:n = { hlines, vlines } ,
8992   hvlines .value_forbidden:n = true ,
8993   name .code:n =
8994     \tl_if_empty:nTF { #1 }
8995     { \@@_error:n { Invalid-name } }
8996     {
8997       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8998       {
8999         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9000         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9001         {
9002           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9003           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9004         }
9005       }
9006       { \@@_error:n { Invalid-name } }
9007     } ,
9008   name .value_required:n = true ,
9009   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9010   rules .value_required:n = true ,
9011   code .tl_set:N = \l_@@_code_tl ,
9012   code .value_required:n = true ,
9013   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9014 }

```

```

9015 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
9016 {
9017   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9018   {
9019     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9020     [
9021       delimiters / color = \l_@@_delimiters_color_tl ,
9022       hlines = \l_@@_submatrix_hlines_clist ,
9023       vlines = \l_@@_submatrix_vlines_clist ,
9024       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9025       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9026       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9027       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9028       #5
9029     ]
9030   }
9031   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9032   \ignorespaces
9033 }
9034 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9035 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9036 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9037 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9038 {
9039   \seq_gput_right:Ne \g_@@_submatrix_seq
9040   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9041   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9042   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9043   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9044   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9045 }
9046 }

```

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).

```

9047 \NewDocumentCommand \@@_compute_i_j:nn
9048 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9049 { \@@_compute_i_j:nnnn #1 #2 }
9050 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9051 {
9052   \def \l_@@_first_i_tl { #1 }
9053   \def \l_@@_first_j_tl { #2 }
9054   \def \l_@@_last_i_tl { #3 }
9055   \def \l_@@_last_j_tl { #4 }
9056   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9057   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9058   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9059   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9060   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9061   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9062   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9063   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9064 }

```

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`.

```

9065 \hook_gput_code:nnn { begindocument } { . }
9066 {
9067   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9068   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9069     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9070 }
9071 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9072 {
9073   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9074   \@@_compute_i_j:nn { #2 } { #3 }
9075   \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
9076   { \def \arraystretch { 1 } }
9077   \bool_lazy_or:nnTF
9078     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9079     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9080     { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9081   {
9082     \str_clear_new:N \l_@@_submatrix_name_str
9083     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9084     \pgfpicture
9085     \pgfrememberpicturepositiononpagetrue
9086     \pgf@relevantforpicturesizefalse
9087     \pgfset { inner~sep = \c_zero_dim }
9088     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9089     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9090   \bool_if:NTF \l_@@_submatrix_slim_bool
9091   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
9092   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
9093   {
9094     \cs_if_exist:cT
9095     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9096     {
9097       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9098       \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9099       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9100     }
9101     \cs_if_exist:cT
9102     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9103     {
9104       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9105       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9106       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9107     }
9108   }
9109   \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
9110   { \@@_error:nn { Impossible~delimiter } { left } }
9111   {
9112     \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

9113         { \@@_error:nn { Impossible-delimiter } { right } }
9114         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9115     }
9116     \endpgfpicture
9117 }
9118 \group_end:
9119 \ignorespaces
9120 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9121 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9122 {
9123     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9124     \dim_set:Nn \l_@@_y_initial_dim
9125     {
9126         \fp_to_dim:n
9127         {
9128             \pgf@y
9129             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9130         }
9131     }
9132     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9133     \dim_set:Nn \l_@@_y_final_dim
9134     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9135     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9136     {
9137         \cs_if_exist:cT
9138         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9139         {
9140             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9141             \dim_set:Nn \l_@@_y_initial_dim
9142             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9143         }
9144         \cs_if_exist:cT
9145         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9146         {
9147             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9148             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9149             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9150         }
9151     }
9152     \dim_set:Nn \l_tmpa_dim
9153     {
9154         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9155         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9156     }
9157     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9158     \group_begin:
9159     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9160     \@@_set_CTarc:o \l_@@_rules_color_tl
9161     \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`.

```

9162     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9163     {
9164         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9165         {
9166             \int_compare:nNnT
9167             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```

9168         {
First, we extract the value of the abscissa of the rule we have to draw.
9169         \@_qpoint:n { col - ##1 }
9170         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9171         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9172         \pgfusepathqstroke
9173     }
9174 }
9175 }

```

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.

```

9176 \str_if_eq:eeTF { \l_@_submatrix_vlines_clist } { all }
9177 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
9178 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
9179 {
9180     \bool_lazy_and:nnTF
9181     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9182     {
9183         \int_compare_p:nNn
9184         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
9185     {
9186         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
9187         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9188         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9189         \pgfusepathqstroke
9190     }
9191     { \@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9192 }

```

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.

```

9193 \str_if_eq:eeTF { \l_@_submatrix_hlines_clist } { all }
9194 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
9195 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
9196 {
9197     \bool_lazy_and:nnTF
9198     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9199     {
9200         \int_compare_p:nNn
9201         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
9202     {
9203         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9204     \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
9205     \dim_set:Nn \l_tmpa_dim
9206     { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
9207     \str_case:nn { #1 }
9208     {
9209         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9210         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9211         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9212     }
9213     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

9214     \dim_set:Nn \l_tmpb_dim
9215     { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
9216     \str_case:nn { #2 }
9217     {
9218         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

```

```

9219         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9220         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9221     }
9222     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9223     \pgfusepathqstroke
9224     \group_end:
9225 }
9226 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9227 }

```

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).

```

9228 \str_if_empty:NF \l_@@_submatrix_name_str
9229 {
9230     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9231     \l_@@_x_initial_dim \l_@@_y_initial_dim
9232     \l_@@_x_final_dim \l_@@_y_final_dim
9233 }
9234 \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`.

```

9235 \begin { pgfscope }
9236 \pgftransformshift
9237 {
9238     \pgfpoint
9239     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9240     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9241 }
9242 \str_if_empty:NTF \l_@@_submatrix_name_str
9243 { \@@_node_left:nn #1 { } }
9244 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9245 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9246 \pgftransformshift
9247 {
9248     \pgfpoint
9249     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9250     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9251 }
9252 \str_if_empty:NTF \l_@@_submatrix_name_str
9253 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9254 {
9255     \@@_node_right:nnnn #2
9256     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9257 }

```

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`.

```

9258 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9259 \flag_clear_new:N \l_@@_code_flag
9260 \l_@@_code_tl
9261 }

```

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`.

```

9262 \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.

```
9263 \cs_new:Npn \@@_pgfpointanchor:n #1
9264 { \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`.

```
9265 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9266 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9267 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9268 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9269 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9270 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9271 { \@@_pgfpointanchor_ii:n { #1 } }
9272 }
```

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`).

```
9273 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9274 { \@@_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`.

```
9275 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9276 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9277 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9278 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9279 { \@@_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`).

```
9280 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9281 }
```

The following function is for the case when the name contains an hyphen.

```
9282 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9283 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9284 \@@_env:
9285 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9286 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9287 }
```



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.

```

9288 \tl_const:Nn \c_@@_integers_alist_tl
9289 {
9290   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9291   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9292   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9293   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9294 }

9295 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9296 {

```

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`.

```

9297   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9298   {
9299     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9300   \@@_env: -
9301   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9302   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9303   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9304 }
9305 {
9306   \str_if_eq:eeTF { #1 } { last }
9307   {
9308     \flag_raise:N \l_@@_code_flag
9309     \@@_env: -
9310     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9311     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9312     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9313   }
9314   { #1 }
9315 }
9316 }

```

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`).

```

9317 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9318 {
9319   \pgfnode
9320   { rectangle }
9321   { east }
9322   {
9323     \nullfont
9324     $ % $
9325     \@@_color:o \l_@@_delimiters_color_tl
9326     \left #1
9327     \vcenter
9328     {
9329       \nullfont
9330       \hrule \@height \l_tmpa_dim
9331       \@depth \c_zero_dim

```

```

9332         \@width \c_zero_dim
9333     }
9334     \right .
9335     $ % $
9336 }
9337 { #2 }
9338 { }
9339 }

```

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.

```

9340 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9341 {
9342     \pgfnode
9343     { rectangle }
9344     { west }
9345     {
9346         \nullfont
9347         $ % $
9348         \colorlet { current-color } { . }
9349         \@@_color:o \l_@@_delimiters_color_tl
9350         \left .
9351         \vcenter
9352         {
9353             \nullfont
9354             \hrule \@height \l_tmpa_dim
9355                 \@depth \c_zero_dim
9356                 \@width \c_zero_dim
9357         }
9358         \right #1
9359         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9360         ^ { \color { current-color } \smash { #4 } }
9361         $ % $
9362     }
9363     { #2 }
9364     { }
9365 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9366 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9367 {
9368     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9369     \ignorespaces
9370 }
9371 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9372 {
9373     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9374     \ignorespaces
9375 }
9376 \keys_define:nn { nicematrix / Brace }
9377 {
9378     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9379     left-shorten .default:n = true ,
9380     left-shorten .value_forbidden:n = true ,

```

```

9381 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9382 right-shorten .default:n = true ,
9383 right-shorten .value_forbidden:n = true ,
9384 shorten .meta:n = { left-shorten , right-shorten } ,
9385 shorten .value_forbidden:n = true ,
9386 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9387 yshift .value_required:n = true ,
9388 yshift .initial:n = \c_zero_dim ,
9389 color .tl_set:N = \l_tmpa_tl ,
9390 color .value_required:n = true ,
9391 unknown .code:n =
9392   \@@_unknown_key:nn
9393     { nicematrix / Brace }
9394     { Unknown-key-for-Brace }
9395 }

```

#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*.

```

9396 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9397 {
9398   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9399   \@@_compute_i_j:nn { #1 } { #2 }
9400   \bool_lazy_or:nnTF
9401     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9402     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9403     {
9404       \str_if_eq:eeTF { #5 } { under }
9405       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9406       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9407     }
9408     {
9409       \tl_clear:N \l_tmpa_tl
9410       \keys_set:nn { nicematrix / Brace } { #4 }
9411       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9412       \pgfpicture
9413       \pgfrememberpicturepositiononpagetrue
9414       \pgf@relevantforpicturesizefalse
9415       \bool_if:NT \l_@@_brace_left_shorten_bool
9416       {
9417         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9418         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9419         {
9420           \cs_if_exist:cT
9421             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9422             {
9423               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9424
9425               \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9426               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9427             }
9428         }
9429       }
9430       \bool_lazy_or:nnT
9431         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9432         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9433         {
9434           \@@_qpoint:n { col - \l_@@_first_j_tl }
9435           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9436         }
9437       \bool_if:NT \l_@@_brace_right_shorten_bool
9438       {

```

```

9439 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9440 \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9441 {
9442   \cs_if_exist:cT
9443   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9444   {
9445     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9446     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9447     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9448   }
9449 }
9450 }
9451 \bool_lazy_or:nnT
9452 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9453 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9454 {
9455   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9456   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9457 }
9458 \pgfset { inner~sep = \c_zero_dim }
9459 \str_if_eq:eeTF { #5 } { under }
9460 { \@@_underbrace_i:n { #3 } }
9461 { \@@_overbrace_i:n { #3 } }
9462 \endpgfpicture
9463 }
9464 \group_end:
9465 }

```

The argument is the text to put above the brace.

```

9466 \cs_new_protected:Npn \@@_overbrace_i:n #1
9467 {
9468   \@@_qpoint:n { row - \l_@@_first_i_tl }
9469   \pgftransformshift
9470   {
9471     \pgfpoint
9472     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9473     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9474   }
9475   \pgfnode
9476   { rectangle }
9477   { south }
9478   {
9479     \vtop
9480     {
9481       \group_begin:
9482       \everycr { }
9483       \halign
9484       {
9485         \hfil ## \hfil \crcr
9486         \bool_if:NTF \l_@@_tabular_bool
9487         { \begin { tabular } { c } #1 \end { tabular } }
9488         { $ \begin { array } { c } #1 \end { array } $ }
9489         \cr
9490         $ % $
9491         \overbrace
9492         {
9493           \hbox_to_wd:nn
9494           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9495           { }
9496         }
9497         $ % $
9498         \cr
9499       }
9500       \group_end:

```

```

9501     }
9502   }
9503   { }
9504   { }
9505 }

```

The argument is the text to put under the brace.

```

9506 \cs_new_protected:Npn \l_@@_underbrace_i:n #1
9507 {
9508   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9509   \pgftransformshift
9510   {
9511     \pgfpoint
9512     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9513     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9514   }
9515   \pgfnode
9516   { rectangle }
9517   { north }
9518   {
9519     \group_begin:
9520     \everycr { }
9521     \vbox
9522     {
9523       \halign
9524       {
9525         \hfil ## \hfil \crcr
9526         $ % $
9527         \underbrace
9528         {
9529           \hbox_to_wd:nn
9530           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9531           { }
9532         }
9533         $ % $
9534         \cr
9535         \bool_if:NTF \l_@@_tabular_bool
9536         { \begin { tabular } { c } #1 \end { tabular } }
9537         { $ \begin { array } { c } #1 \end { array } $ }
9538         \cr
9539       }
9540     }
9541     \group_end:
9542   }
9543   { }
9544   { }
9545 }

```

## 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`).

```

9546 \AddToHook { package / tikz / after }
9547 {

```

```

9548 \tikzset
9549 {
9550     nicematrix / brace / .style =
9551     {
9552         decoration = { brace , raise = -0.15 em } ,
9553         decorate ,
9554     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9555     nicematrix / mirrored-brace / .style =
9556     {
9557         nicematrix / brace ,
9558         decoration = mirror ,
9559     }
9560 }
9561 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9562 \keys_define:nn { nicematrix / Hbrace }
9563 {
9564     color .code:n = ,
9565     horizontal-label .code:n = ,
9566     horizontal-labels .code:n = ,
9567     shorten .code:n = ,
9568     shorten-start .code:n = ,
9569     shorten-end .code:n = ,
9570     shorten+ .code:n = ,
9571     shorten-start+ .code:n = ,
9572     shorten-end+ .code:n = ,
9573     shorten~+ .code:n = ,
9574     shorten-start~+ .code:n = ,
9575     shorten-end~+ .code:n = ,
9576     brace-shift .code:n = ,
9577     brace-shift+ .code:n = ,
9578     brace-shift~+ .code:n = ,
9579     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9580 }

```

Here we need an “fully expandable” command.

```

9581 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9582 {
9583     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9584     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9585     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9586 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9587 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9588 {
9589     \int_compare:nNnTF { \c@iRow } < { 2 }
9590     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9591     \str_if_eq:nnTF { #2 } { * }
9592     {
9593         \bool_set_true:N \l_@@_nullify_dots_bool
9594         \Ldots
9595         [
9596             line-style = nicematrix / brace ,
9597             #1 ,

```

```

9598         up =
9599         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9600     ]
9601 }
9602 {
9603     \Hdotsfor
9604     [
9605         line-style = nicematrix / brace ,
9606         #1 ,
9607         up =
9608         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9609     ]
9610     { #2 }
9611 }
9612 }
9613 {
9614     \str_if_eq:nnTF { #2 } { * }
9615     {
9616         \bool_set_true:N \l_@@_nullify_dots_bool
9617         \Ldots
9618         [
9619             line-style = nicematrix / mirrored-brace ,
9620             #1 ,
9621             down =
9622             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9623         ]
9624     }
9625     {
9626         \Hdotsfor
9627         [
9628             line-style = nicematrix / mirrored-brace ,
9629             #1 ,
9630             down =
9631             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9632         ]
9633         { #2 }
9634     }
9635 }
9636 \keys_set:nn { nicematrix / Hbrace } { #1 }
9637 }

9638 \NewDocumentCommand { \@@_Vbrace } { 0 { } } m m }
9639 {
9640     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9641     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9642     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9643 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9644 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9645 {
9646     \int_compare:nNnTF { \c@jCol } < { 2 }
9647     {
9648         \str_if_eq:nnTF { #2 } { * }
9649         {
9650             \bool_set_true:N \l_@@_nullify_dots_bool
9651             \Vdots
9652             [
9653                 Vbrace ,
9654                 line-style = nicematrix / mirrored-brace ,
9655                 #1 ,
9656                 down =

```

```

9657         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9658     ]
9659 }
9660 {
9661     \Vdotsfor
9662     [
9663         Vbrace ,
9664         line-style = nicematrix / mirrored-brace ,
9665         #1 ,
9666         down =
9667             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9668     ]
9669     { #2 }
9670 }
9671 }
9672 {
9673     \str_if_eq:nnTF { #2 } { * }
9674     {
9675         \bool_set_true:N \l_@@_nullify_dots_bool
9676         \Vdots
9677         [
9678             Vbrace ,
9679             line-style = nicematrix / brace ,
9680             #1 ,
9681             up =
9682                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9683         ]
9684     }
9685     {
9686         \Vdotsfor
9687         [
9688             Vbrace ,
9689             line-style = nicematrix / brace ,
9690             #1 ,
9691             up =
9692                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9693         ]
9694         { #2 }
9695     }
9696 }
9697 \keys_set:nn { nicematrix / Hbrace } { #1 }
9698 }

```

## 36 The command TikzEveryCell

```

9699 \bool_new:N \l_@@_not_empty_bool
9700 \bool_new:N \l_@@_empty_bool
9701
9702 \keys_define:nn { nicematrix / TikzEveryCell }
9703 {
9704     not-empty .code:n =
9705         \bool_lazy_or:nnTF
9706             { \l_@@_in_code_after_bool }
9707             { \g_@@_create_cell_nodes_bool }
9708             { \bool_set_true:N \l_@@_not_empty_bool }
9709             { \@@_error:n { detection-of-empty-cells } } ,
9710     not-empty .value_forbidden:n = true ,
9711     empty .code:n =
9712         \bool_lazy_or:nnTF

```



```

9713     { \l_@@_in_code_after_bool }
9714     { \g_@@_create_cell_nodes_bool }
9715     { \bool_set_true:N \l_@@_empty_bool }
9716     { \@@_error:n { detection-of-empty-cells } } } ,
9717     empty .value_forbidden:n = true ,
9718     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9719 }
9720
9721
9722 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9723 {
9724     \IfPackageLoadedTF { tikz }
9725     {
9726         \group_begin:
9727         \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.

```

9728         \tl_set:Nn \l_tmpa_tl { { #2 } }
9729         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9730         { \@@_for_a_block:nnnnn ##1 }
9731         \@@_all_the_cells:
9732         \group_end:
9733     }
9734     { \@@_error:n { TikzEveryCell-without-tikz } }
9735 }
9736
9737
9738 \cs_new_protected:Nn \@@_all_the_cells:
9739 {
9740     \int_step_inline:nn \c@iRow
9741     {
9742         \int_step_inline:nn \c@jCol
9743         {
9744             \cs_if_exist:cF { cell - ##1 - ####1 }
9745             {
9746                 \clist_if_in:NcF \l_@@_corners_cells_clist
9747                 { ##1 - ####1 }
9748                 {
9749                     \bool_set_false:N \l_tmpa_bool
9750                     \cs_if_exist:cTF
9751                     { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
9752                     {
9753                         \bool_if:NF \l_@@_empty_bool
9754                         { \bool_set_true:N \l_tmpa_bool }
9755                     }
9756                     {
9757                         \bool_if:NF \l_@@_not_empty_bool
9758                         { \bool_set_true:N \l_tmpa_bool }
9759                     }
9760                     \bool_if:NT \l_tmpa_bool
9761                     {
9762                         \@@_block_tikz:nnnnn
9763                         \l_tmpa_tl { ##1 } { ####1 } { ##1 } { ####1 }
9764                     }
9765                 }
9766             }
9767         }
9768     }
9769 }
9770
9771 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9772 {
9773     \bool_if:NF \l_@@_empty_bool

```

```

9774 {
9775   \@@_block_tikz:nnnn
9776   \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9777 }
9778 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9779 }
9780
9781 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9782 {
9783   \int_step_inline:nnn { #1 } { #3 }
9784   {
9785     \int_step_inline:nnn { #2 } { #4 }
9786     { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9787   }
9788 }

```

### 37 The command \ShowCellNames

```

9789 \NewDocumentCommand \@@_ShowCellNames { }
9790 {
9791   \bool_if:NT \l_@@_in_code_after_bool
9792   {
9793     \pgfpicture
9794     \pgfrememberpicturepositiononpagetrue
9795     \pgf@relevantforpicturesizefalse
9796     \pgfpathrectanglecorners
9797     { \@@_qpoint:n { 1 } }
9798     {
9799       \@@_qpoint:n
9800       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9801     }
9802     \pgfsetfillopacity { 0.75 }
9803     \pgfsetfillcolor { white }
9804     \pgfusepathqfill
9805     \endpgfpicture
9806   }
9807   \dim_gzero_new:N \g_@@_tmpc_dim
9808   \dim_gzero_new:N \g_@@_tmpd_dim
9809   \dim_gzero_new:N \g_@@_tmpe_dim
9810   \int_step_inline:nn { \c@iRow }
9811   {
9812     \bool_if:NTF \l_@@_in_code_after_bool
9813     {
9814       \pgfpicture
9815       \pgfrememberpicturepositiononpagetrue
9816       \pgf@relevantforpicturesizefalse
9817     }
9818     { \begin { pgfpicture } }
9819     \@@_qpoint:n { row - ##1 }
9820     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9821     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9822     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9823     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9824     \bool_if:NTF \l_@@_in_code_after_bool
9825     { \endpgfpicture }
9826     { \end { pgfpicture } }
9827     \int_step_inline:nn { \c@jCol }
9828     {
9829       \hbox_set:Nn \l_tmpa_box
9830       {
9831         \normalfont \Large \sffamily \bfseries
9832         \bool_if:NTF \l_@@_in_code_after_bool

```

```

9833         { \color { red } }
9834         { \color { red ! 50 } }
9835     ##1 - ####1
9836 }
9837 \bool_if:NTF \l_@@_in_code_after_bool
9838 {
9839     \pgfpicture
9840     \pgfrememberpicturepositiononpagetrue
9841     \pgf@relevantforpicturesizefalse
9842 }
9843 { \begin { pgfpicture } }
9844 \@@_qpoint:n { col - ####1 }
9845 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9846 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9847 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9848 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9849 \bool_if:NTF \l_@@_in_code_after_bool
9850 { \endpgfpicture }
9851 { \end { pgfpicture } }
9852 \fp_set:Nn \l_tmpa_fp
9853 {
9854     \fp_min:nn
9855     {
9856         \fp_min:nn
9857         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9858         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9859     }
9860     { 1.0 }
9861 }
9862 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9863 \pgfpicture
9864 \pgfrememberpicturepositiononpagetrue
9865 \pgf@relevantforpicturesizefalse
9866 \pgftransformshift
9867 {
9868     \pgfpoint
9869     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9870     { \dim_use:N \g_tmpa_dim }
9871 }
9872 \pgfnode
9873 { rectangle }
9874 { center }
9875 { \box_use:N \l_tmpa_box }
9876 { }
9877 { }
9878 \endpgfpicture
9879 }
9880 }
9881 }

```

## 38 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.

```

9882 \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.

```

9883 \bool_new:N \g_@@_footnote_bool
9884 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9885 {
9886   You-have-used-the-key~' \l_keys_key_str '~when~loading~nicematrix~
9887   but~that~key~is~unknown. \\
9888   It~will~be~ignored. \\
9889   For~a~list~of~the~available~keys,~type~H~<return>.
9890 }
9891 {
9892   The~available~keys~are~(in~alphabetic~order):~
9893   footnote,~
9894   footnotehyper,~
9895   messages-for-Overleaf,~
9896   renew-dots~and~
9897   renew-matrix.
9898 }
9899 \keys_define:nn { nicematrix }
9900 {
9901   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9902   renew-dots .value_forbidden:n = true ,
9903   renew-matrix .code:n = \@@_renew_matrix: ,
9904   renew-matrix .value_forbidden:n = true ,
9905   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9906   footnote .bool_set:N = \g_@@_footnote_bool ,
9907   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9908   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9909 }
9910 \ProcessKeyOptions
9911 \@@_msg_new:nn { footnote-with-footnotehyper~package }
9912 {
9913   You~can't~use~the~option~'footnote'~because~the~package~
9914   footnotehyper~has~already~been~loaded.~
9915   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9916   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9917   of~the~package~footnotehyper.\\
9918   The~package~footnote~won't~be~loaded.
9919 }
9920 \@@_msg_new:nn { footnotehyper-with-footnote~package }
9921 {
9922   You~can't~use~the~option~'footnotehyper'~because~the~package~
9923   footnote~has~already~been~loaded.~
9924   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9925   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9926   of~the~package~footnote.\\
9927   The~package~footnotehyper~won't~be~loaded.
9928 }
9929 \bool_if:NT \g_@@_footnote_bool
9930 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9931 \IfClassLoadedTF { beamer }
9932 { \bool_set_false:N \g_@@_footnote_bool }
9933 {
9934   \IfPackageLoadedTF { footnotehyper }
9935   { \@@_error:n { footnote-with-footnotehyper~package } }
9936   { \usepackage { footnote } }

```

```

9937     }
9938 }
9939 \bool_if:NT \g_@@_footnotehyper_bool
9940 {

```

The class `beamer` has its own system to extract footnotes and that’s why we have nothing to do if `beamer` is used.

```

9941 \IfClassLoadedTF { beamer }
9942 { \bool_set_false:N \g_@@_footnote_bool }
9943 {
9944     \IfPackageLoadedTF { footnote }
9945     { \@@_error:n { footnotehyper~with~footnote~package } }
9946     { \usepackage { footnotehyper } }
9947 }
9948 \bool_set_true:N \g_@@_footnote_bool
9949 }

```

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}`.

## 39 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.

```

9950 \bool_new:N \l_@@_underscore_loaded_bool
9951 \IfPackageLoadedT { underscore }
9952 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9953 \hook_gput_code:nnn { begindocument } { . }
9954 {
9955     \bool_if:NF \l_@@_underscore_loaded_bool
9956     {
9957         \IfPackageLoadedT { underscore }
9958         { \@@_error:n { underscore~after~nicematrix } }
9959     }
9960 }

```

## 40 Error messages of the package

When there is a unknown key, we try a “normal form” 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.

```

9961 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
9962 {
9963     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
9964     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
9965     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
9966     \bool_set_false:N \l_tmpa_bool
9967     \clist_map_inline:nn { #1 }
9968     {
9969         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
9970         {

```

```

9971         \@@_error:n { key-with-normal-form-exists }
9972         \bool_set_true:N \l_tmpa_bool
9973         \clist_map_break:
9974     }
9975 }
9976 \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
9977 }
9978 \@@_msg_new:nn { key-with-normal-form-exists }
9979 {
9980     The-key~'\l_keys_key_str'~does-not-exists.\\
9981     It~will~be-ignored.\\
9982     Maybe-you-want-to-use-the-key~'\l_tmpa_str'.
9983 }
9984 \str_const:Ne \c_@@_available_keys_str
9985 {
9986     \bool_if:Nt { ! \g_@@_messages_for_Overleaf_bool }
9987     { For~a~list-of-the~available~keys,~type-H~<return>. }
9988 }
9989 \seq_new:N \g_@@_types_of_matrix_seq
9990 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9991 {
9992     NiceMatrix ,
9993     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9994 }
9995 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9996 { \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`.

```

9997 \cs_new_protected:Npn \@@_err_too_many_cols:
9998 {
9999     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10000     { \@@_fatal:nn { too-many-cols-for-array } }
10001     \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
10002     { \@@_fatal:n { too-many-cols-for-matrix } }
10003     \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
10004     { \@@_fatal:n { too-many-cols-for-matrix } }
10005     \bool_if:NF \l_@@_last_col_without_value_bool
10006     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10007 }

```

The following command must *not* be protected since it's used in an error message.

```

10008 \cs_new:Npn \@@_message_hdotsfor:
10009 {
10010     \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
10011     { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10012       \token_to_str:N \Hbrace \ is~incorrect. }
10013 }
10014 \cs_new_protected:Npn \@@_Hline_in_cell:
10015 { \@@_fatal:n { Misuse-of-Hline } }
10016 \@@_msg_new:nn { Misuse-of-Hline }
10017 {
10018     Misuse-of-Hline. \\
10019     Error-in-your-row~ \int_eval:n { \c@iRow }. \\
10020     \token_to_str:N \Hline\ must-be-used-only-at-the-beginning-of-a-row.\\
10021     That-error-is-fatal.
10022 }

```

```

10023 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10024 {
10025     Incompatible-options.\\
10026     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
10027     The-output~will~not~be~reliable.
10028 }
10029 \@@_msg_new:nn { Body-alone }
10030 {
10031     \token_to_str:N \Body\ alone. \\
10032     You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10033     That~error~is~fatal.
10034 }
10035 \@@_msg_new:nn { cellcolor-in-Block }
10036 {
10037     Bad-use-of~\token_to_str:N \cellcolor \\
10038     You-can't-use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10039     (except~in~a~sub-block).\\
10040     That~command~will~be~ignored.
10041 }
10042 \@@_msg_new:nn { rowcolor-in-Block }
10043 {
10044     Bad-use-of~\token_to_str:N \rowcolor \\
10045     You-can't-use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10046     That~command~will~be~ignored.
10047 }
10048 \@@_msg_new:nn { key~color~inside }
10049 {
10050     Deleted-key.\\
10051     The-key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10052     ~'nicematrix'~and~must~not~be~used.\\
10053     This~error~is~fatal.
10054 }
10055 \@@_msg_new:nn { invalid-weight }
10056 {
10057     Unknown~key.\\
10058     The-key~'\l_keys_key_str '~-of~your~column~X~is~unknown~and~will~be~ignored.
10059 }
10060 \@@_msg_new:nn { last-col-not-used }
10061 {
10062     Column~not~used.\\
10063     The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
10064     in~your~\@@_full_name_env: .~
10065     However,~you~can~go~on.
10066 }
10067 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10068 {
10069     Too-many-columns.\\
10070     In~the~row~\int_eval:n { \c@iRow },~
10071     you~try~to~use~more~columns~
10072     than~allowed~by~your~\@@_full_name_env: .
10073     \@@_message_hdotsfor: \
10074     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
10075     (plus~the~exterior~columns).~This~error~is~fatal.
10076 }
10077 \@@_msg_new:nn { too-many-cols-for-matrix }
10078 {
10079     Too-many-columns.\\
10080     In~the~row~\int_eval:n { \c@iRow },~
10081     you~try~to~use~more~columns~than~allowed~by~your~\@@_full_name_env: .
10082     \@@_message_hdotsfor: \

```

```

10083 Recall~that~the~maximal~number~of~columns~for~a~matrix~
10084 (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10085 LaTeX~counter~'MaxMatrixCols'.~
10086 Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10087 (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10088 This~error~is~fatal.
10089 }

10090 \@@_msg_new:nn { too-many-cols-for-array }
10091 {
10092   Too-many-columns.\\
10093   In~the~row~ \int_eval:n { \c@iRow } ,~
10094   ~you~try~to~use~more~columns~than~allowed~by~your~
10095   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10096   \int_use:N \g_@@_static_num_of_col_int \
10097   \bool_if:nT
10098     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10099     { (plus~the~exterior~ones)~}
10100   since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
10101   This~error~is~fatal.
10102 }

10103 \@@_msg_new:nn { columns-not-used }
10104 {
10105   Columns-not-used.\\
10106   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
10107   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10108   columns~but~you~only~used~ \int_use:N \c@jCol .\\
10109   The~columns~you~did~not~used~won't~be~created.\\
10110   You~won't~have~similar~warning~till~the~end~of~the~document.
10111 }

10112 \@@_msg_new:nn { empty-preamble }
10113 {
10114   Empty-preamble.\\
10115   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10116   This~error~is~fatal.
10117 }

10118 \@@_msg_new:nn { in-first-col }
10119 {
10120   Erroneous-use.\\
10121   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
10122   That~command~will~be~ignored.
10123 }

10124 \@@_msg_new:nn { in-last-col }
10125 {
10126   Erroneous-use.\\
10127   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
10128   That~command~will~be~ignored.
10129 }

10130 \@@_msg_new:nn { in-first-row }
10131 {
10132   Erroneous-use.\\
10133   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10134   That~command~will~be~ignored.
10135 }

10136 \@@_msg_new:nn { in-last-row }
10137 {
10138   Erroneous-use.\\
10139   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10140   That~command~will~be~ignored.
10141 }

```



```

10142 \@@_msg_new:nn { TopRule~without~booktabs }
10143 {
10144   Erroneous~use.\\
10145   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10146   That~command~will~be~ignored.
10147 }
10148 \@@_msg_new:nn { TopRule~without~tikz }
10149 {
10150   Erroneous~use.\\
10151   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
10152   That~command~will~be~ignored.
10153 }
10154 \@@_msg_new:nn { caption~outside~float }
10155 {
10156   Key~caption~forbidden.\\
10157   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10158   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10159 }
10160 \@@_msg_new:nn { short~caption~without~caption }
10161 {
10162   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10163   However,~your~'short~caption'~will~be~used~as~'caption'.
10164 }
10165 \@@_msg_new:nn { double~closing~delimiter }
10166 {
10167   Double~delimiter.\\
10168   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10169   delimiter.~This~delimiter~will~be~ignored.
10170 }
10171 \@@_msg_new:nn { delimiter~after~opening }
10172 {
10173   Double~delimiter.\\
10174   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10175   delimiter.~That~delimiter~will~be~ignored.
10176 }
10177 \@@_msg_new:nn { bad~option~for~line~style }
10178 {
10179   Bad~line~style.\\
10180   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10181   is~'standard'.~That~key~will~be~ignored.
10182 }
10183 \@@_msg_new:nn { corners~with~no~cell~nodes }
10184 {
10185   Incompatible~keys.\\
10186   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10187   is~in~force.\\
10188   If~you~go~on,~that~key~will~be~ignored.
10189 }
10190 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10191 {
10192   Incompatible~keys.\\
10193   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10194   is~in~force.\\
10195   If~you~go~on,~those~extra~nodes~won't~be~created.
10196 }
10197 \@@_msg_new:nn { Identical~notes~in~caption }
10198 {
10199   Identical~tabular~notes.\\
10200   You~can't~put~several~notes~with~the~same~content~in~
10201   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\

```

```

10202     If-you-go-on,~the-output-will-probably-be-erroneous.
10203 }
10204 \@@_msg_new:nn { tabularnote~below~the~tabular }
10205 {
10206     \token_to_str:N \tabularnote \ forbidden\\
10207     You-can't-use~ \token_to_str:N \tabularnote \ in~the~caption~
10208     of~your~tabular~because~the~caption~will~be~composed~below~
10209     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10210     key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10211     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10212     no~similar~error~will~raised~in~this~document.
10213 }
10214 \@@_msg_new:nn { Unknown~key~for~rules }
10215 {
10216     Unknown~key.\\
10217     There~is~only~two~keys~available~here:~width~and~color.\\
10218     Your~key~' \l_keys_key_str '~will~be~ignored.
10219 }
10220 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10221 {
10222     Unknown~key.\\
10223     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10224     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10225     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10226     'horizontal-label(s)',~'shorten'~'shorten-end'~
10227     and~'shorten-start'.\\
10228     That~error~is~fatal.
10229 }
10230 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10231 {
10232     Unknown~key.\\
10233     There~is~only~two~keys~available~here:~
10234     'empty'~and~'not-empty'.\\
10235     Your~key~' \l_keys_key_str '~will~be~ignored.
10236 }
10237 \@@_msg_new:nn { Unknown~key~for~rotate }
10238 {
10239     Unknown~key.\\
10240     The~only~key~available~here~is~'c'.\\
10241     Your~key~' \l_keys_key_str '~will~be~ignored.
10242 }
10243 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10244 {
10245     Unknown~key.\\
10246     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10247     It~you~go~on,~you~will~probably~have~other~errors. \\
10248     \c_@@_available_keys_str
10249 }
10250 {
10251     The~available~keys~are~(in~alphabetic~order):~
10252     ccommand,~
10253     color,~
10254     command,~
10255     dotted,~
10256     letter,~
10257     multiplicity,~
10258     sep-color,~
10259     tikz,~and~total-width.
10260 }
10261 \@@_msg_new:nnn { Unknown~key~for~xdots }
10262 {

```

```

10263     Unknown~key.\\
10264     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10265     \c_@@_available_keys_str
10266 }
10267 {
10268     The~available~keys~are~(in~alphabetic~order):~
10269     'color',~
10270     'horizontal(s)-labels',~
10271     'inter',~
10272     'line-style',~
10273     'nullify',~
10274     'radius',~
10275     'shorten',~
10276     'shorten-end'~and~'shorten-start'.
10277 }
10278 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10279 {
10280     Unknown~key.\\
10281     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10282     (and~you~try~to~use~' \l_keys_key_str '~)\\
10283     That~key~will~be~ignored.
10284 }
10285 \@@_msg_new:nn { label~without~caption }
10286 {
10287     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10288     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10289 }
10290 \@@_msg_new:nn { W~warning }
10291 {
10292     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10293     (row~ \int_use:N \c@iRow ).
10294 }
10295 \@@_msg_new:nn { Construct~too~large }
10296 {
10297     Construct~too~large.\\
10298     Your~command~ \token_to_str:N #1
10299     can't~be~drawn~because~your~matrix~is~too~small.\\
10300     That~command~will~be~ignored.
10301 }
10302 \@@_msg_new:nn { underscore~after~nicematrix }
10303 {
10304     Problem~with~'underscore'.\\
10305     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10306     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10307     ' \token_to_str:N \Cdots \token_to_str:N _
10308     \{ n \token_to_str:N \text \{ ~times \} \}' .
10309 }
10310 \@@_msg_new:nn { ampersand~in~light~syntax }
10311 {
10312     Ampersand~forbidden.\\
10313     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10314     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10315 }
10316 \@@_msg_new:nn { double~backslash~in~light~syntax }
10317 {
10318     Double~backslash~forbidden.\\
10319     You~can't~use~ \token_to_str:N \\
10320     ~to~separate~rows~because~the~key~'light-syntax'~
10321     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10322     (set~by~the~key~'end-of-row').~This~error~is~fatal.
10323 }

```

```

10324 \@@_msg_new:nn { hlines-with-color }
10325 {
10326   Incompatible-keys.\
10327   You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for~a~
10328   \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\
10329   However,~you~can~put~several~commands~ \token_to_str:N \Block.\
10330   Your~key~will~be~discarded.
10331 }
10332 \@@_msg_new:nn { bad-value-for-baseline }
10333 {
10334   Bad-value-for-baseline.\
10335   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10336   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10337   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10338   the~form~'line-i'.\
10339   A~value~of~1~will~be~used.
10340 }
10341 \@@_msg_new:nn { bad-value-for-baseline-line }
10342 {
10343   Bad-value-for-baseline-with-line.\
10344   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10345   valid.~The~number~of~the~line~must~be~between~1~and~
10346   \int_eval:n { \c@iRow + 1 } \
10347   A~value~of~'line-1'~will~be~used.
10348 }
10349 \@@_msg_new:nn { detection-of-empty-cells }
10350 {
10351   Problem-with-'not-empty'\
10352   For~technical~reasons,~you~must~activate~
10353   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10354   in~order~to~use~the~key~' \l_keys_key_str '.\
10355   That~key~will~be~ignored.
10356 }
10357 \@@_msg_new:nn { siunitx-not-loaded }
10358 {
10359   siunitx-not-loaded\
10360   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\
10361   That~error~is~fatal.
10362 }
10363 \@@_msg_new:nn { Invalid-name }
10364 {
10365   Invalid-name.\
10366   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10367   \SubMatrix \ of~your~ \@@_full_name_env: .\
10368   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
10369   This~key~will~be~ignored.
10370 }
10371 \@@_msg_new:nn { Hbrace-not-allowed }
10372 {
10373   Command-not-allowed.\
10374   You~can't~use~the~command~ \token_to_str:N #1
10375   because~you~have~not~loaded~
10376   \IfPackageLoadedTF { tikz }
10377   { the~TikZ-library~'decorations.pathreplacing'~.~Use~ }
10378   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10379   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10380   That~command~will~be~ignored.
10381 }
10382 \@@_msg_new:nn { Vbrace-not-allowed }
10383 {
10384   Command-not-allowed.\

```

```

10385     You~can't~use~the~command~ \token_to_str:N \Vbrace \
10386     because~you~have~not~loaded~TikZ~
10387     and~the~TikZ~library~'decorations.pathreplacing'.\\
10388     Use: ~\token_to_str:N \usepackage \{tikz\}~
10389     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10390     That~command~will~be~ignored.
10391   }
10392   \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10393   {
10394     Wrong~line.\\
10395     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10396     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10397     number~is~not~valid.~It~will~be~ignored.
10398   }
10399   \@@_msg_new:nn { Impossible~delimiter }
10400   {
10401     Impossible~delimiter.\\
10402     It's~impossible~to~draw~the~#1~delimiter~of~your~
10403     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10404     in~that~column.
10405     \bool_if:NT \l_@@_submatrix_slim_bool
10406     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10407     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10408   }
10409   \@@_msg_new:nnn { width~without~X~columns }
10410   {
10411     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10412     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10413     That~key~will~be~ignored.
10414   }
10415   {
10416     This~message~is~the~message~'width~without~X~columns'~
10417     of~the~module~'nicematrix'.~
10418     The~experimented~users~can~disable~that~message~with~
10419     \token_to_str:N \msg_redirect_name:nnn .\\
10420   }
10421
10422   \@@_msg_new:nn { key~multiplicity~with~dotted }
10423   {
10424     Incompatible~keys. \\
10425     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10426     in~a~'custom~line'.~They~are~incompatible. \\
10427     The~key~'multiplicity'~will~be~discarded.
10428   }
10429   \@@_msg_new:nn { empty~environment }
10430   {
10431     Empty~environment.\\
10432     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10433   }
10434   \@@_msg_new:nn { No~letter~and~no~command }
10435   {
10436     Erroneous~use.\\
10437     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10438     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10439     '~ccommand'~(to~draw~horizontal~rules).\\
10440     However,~you~can~go~on.
10441   }
10442   \@@_msg_new:nn { Forbidden~letter }
10443   {
10444     Forbidden~letter.\\
10445     You~can't~use~the~letter~'#1'~for~a~customized~line.~

```

```

10446     It~will~be~ignored.\\
10447     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10448 }
10449 \@@_msg_new:nn { Several~letters }
10450 {
10451     Wrong~name.\\
10452     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10453     have~used~' \l_@@_letter_str ').\\
10454     It~will~be~ignored.
10455 }
10456 \@@_msg_new:nn { Delimiter~with~small }
10457 {
10458     Delimiter~forbidden.\\
10459     You~can't~put~a~delimiter~in~the~preamble~of~your~
10460     \@@_full_name_env: \
10461     because~the~key~'small'~is~in~force.\\
10462     This~error~is~fatal.
10463 }
10464 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10465 {
10466     Unknown~cell.\\
10467     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10468     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10469     can't~be~executed~because~a~cell~doesn't~exist.\\
10470     This~command~ \token_to_str:N \line \ will~be~ignored.
10471 }
10472 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10473 {
10474     Duplicate~name.\\
10475     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10476     in~this~ \@@_full_name_env: .\\
10477     This~key~will~be~ignored.\\
10478     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10479     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10480 }
10481 {
10482     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10483     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10484 }
10485 \@@_msg_new:nn { r~or~l~with~preamble }
10486 {
10487     Erroneous~use.\\
10488     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10489     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10490     your~ \@@_full_name_env: .\\
10491     This~key~will~be~ignored.
10492 }
10493 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10494 {
10495     Erroneous~use.\\
10496     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10497     in~an~exterior~column~of~
10498     the~array.~This~error~is~fatal.
10499 }
10500 \@@_msg_new:nn { bad~corner }
10501 {
10502     Bad~corner.\\
10503     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10504     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10505     This~specification~of~corner~will~be~ignored.
10506 }

```

```

10507 \@@_msg_new:nn { bad-border }
10508 {
10509   Bad-border.\
10510   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10511   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10512   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10513   also~use~the~key~'tikz'
10514   \IfPackageLoadedF { tikz }
10515     { ~if~you~load~the~LaTeX~package~'tikz' } ).\
10516   This~specification~of~border~will~be~ignored.
10517 }
10518 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10519 {
10520   TikZ~not~loaded.\
10521   You~can't~use~ \token_to_str:N \TikzEveryCell \
10522   because~you~have~not~loaded~tikz.~
10523   This~command~will~be~ignored.
10524 }
10525 \@@_msg_new:nn { tikz-key~without~tikz }
10526 {
10527   TikZ~not~loaded.\
10528   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10529   \Block '~because~you~have~not~loaded~tikz.~
10530   This~key~will~be~ignored.
10531 }
10532 \@@_msg_new:nn { Bad~argument~for~Block }
10533 {
10534   Bad~argument.\
10535   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10536   be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10537   '#1'. \
10538   If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
10539   the~argument~was~empty).
10540 }
10541 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
10542 {
10543   Erroneous~use.\
10544   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10545   'last-col'~without~value.\
10546   However,~you~can~go~on~for~this~time~
10547   (the~value~' \l_keys_value_tl '~will~be~ignored).
10548 }
10549 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
10550 {
10551   Erroneous~use. \
10552   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10553   'last-col'~without~value. \
10554   However,~you~can~go~on~for~this~time~
10555   (the~value~' \l_keys_value_tl '~will~be~ignored).
10556 }
10557 \@@_msg_new:nn { Block~too~large~1 }
10558 {
10559   Block~too~large. \
10560   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10561   too~small~for~that~block. \
10562   This~block~and~maybe~others~will~be~ignored.
10563 }
10564 \@@_msg_new:nn { Block~too~large~2 }
10565 {
10566   Block~too~large. \
10567   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N

```

```

10568 \g_@@_static_num_of_col_int \
10569 columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10570 specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
10571 (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\\
10572 This~block~and~maybe~others~will~be~ignored.
10573 }
10574 \@@_msg_new:nn { unknown~column~type }
10575 {
10576   Bad~column~type. \\\
10577   The~column~type~'#1'~in~your~ \@@_full_name_env: \
10578   is~unknown. \\\
10579   This~error~is~fatal.
10580 }
10581 \@@_msg_new:nn { unknown~column~type~multicolumn }
10582 {
10583   Bad~column~type. \\\
10584   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10585   ~of~your~ \@@_full_name_env: \
10586   is~unknown. \\\
10587   This~error~is~fatal.
10588 }
10589 \@@_msg_new:nn { unknown~column~type~S }
10590 {
10591   Bad~column~type. \\\
10592   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\\
10593   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10594   load~that~package. \\\
10595   This~error~is~fatal.
10596 }
10597 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
10598 {
10599   Bad~column~type. \\\
10600   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10601   of~your~ \@@_full_name_env: \ is~unknown. \\\
10602   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10603   load~that~package. \\\
10604   This~error~is~fatal.
10605 }
10606 \@@_msg_new:nn { tabularnote~forbidden }
10607 {
10608   Forbidden~command. \\\
10609   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10610   ~here.~This~command~is~available~only~in~
10611   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10612   the~argument~of~a~command~\token_to_str:N \caption \ included~
10613   in~an~environment~\{table\}. \\\
10614   This~command~will~be~ignored.
10615 }
10616 \@@_msg_new:nn { borders~forbidden }
10617 {
10618   Forbidden~key.\\
10619   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10620   because~the~option~'rounded~corners'~
10621   is~in~force~with~a~non~zero~value.\\
10622   This~key~will~be~ignored.
10623 }
10624 \@@_msg_new:nn { bottomrule~without~booktabs }
10625 {
10626   booktabs~not~loaded.\\
10627   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10628   loaded~'booktabs'.\\

```



```

10629     This~key~will~be~ignored.
10630 }
10631 \@@_msg_new:nn { enumitem~not~loaded }
10632 {
10633     enumitem~not~loaded. \\
10634     You~can't~use~the~command~ \token_to_str:N \tabularnote \
10635     ~because~you~haven't~loaded~'enumitem'. \\
10636     All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10637     ignored~in~the~document.
10638 }
10639 \@@_msg_new:nn { tikz~without~tikz }
10640 {
10641     TikZ~not~loaded. \\
10642     You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~
10643     loaded.~If~you~go~on,~that~key~will~be~ignored.
10644 }
10645 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10646 {
10647     TikZ~not~loaded. \\
10648     You~have~used~the~key~'tikz'~in~the~definition~of~a~
10649     customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
10650     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10651     use~that~custom~line.
10652 }
10653 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10654 {
10655     TikZ~not~loaded. \\
10656     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10657     command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
10658     That~key~will~be~ignored.
10659 }
10660 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10661 {
10662     Erroneous~use.\\
10663     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10664     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10665     The~key~'color'~will~be~discarded.
10666 }
10667 \@@_msg_new:nn { Wrong~last~row }
10668 {
10669     Wrong~number.\\
10670     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '-but~your~
10671     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10672     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10673     last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10674     problem~by~using~'last~row'~without~value~(more~compilations~
10675     might~be~necessary).
10676 }
10677 \@@_msg_new:nn { Yet~in~env }
10678 {
10679     Nested~environments.\\
10680     Environments~of~nicematrix~can't~be~nested.\\
10681     This~error~is~fatal.
10682 }
10683 \@@_msg_new:nn { Outside~math~mode }
10684 {
10685     Outside~math~mode.\\
10686     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10687     (and~not~in~ \token_to_str:N \vcenter ).\\
10688     This~error~is~fatal.
10689 }

```

```

10690 \@@_msg_new:nn { One-letter-allowed }
10691 {
10692   Bad-name.\
10693   The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10694   you-have-used~' \l_keys_value_tl '~.\
10695   It~will~be~ignored.
10696 }
10697 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10698 {
10699   Environment~\{TabularNote\}~forbidden.\
10700   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10701   but~*before*~the~ \token_to_str:N \CodeAfter . \
10702   This~environment~\{TabularNote\}~will~be~ignored.
10703 }
10704 \@@_msg_new:nn { varwidth-not-loaded }
10705 {
10706   varwidth-not-loaded.\
10707   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10708   loaded.\
10709   Your~column~will~behave~like~'p'.
10710 }
10711 \@@_msg_new:nn { varwidth-not-loaded-in-X }
10712 {
10713   varwidth-not-loaded.\
10714   You~can't~use~the~key~'V'~in~your~column~'X'~
10715   because~'varwidth'~is~not~loaded.\
10716   It~will~be~ignored. \
10717 }
10718 \@@_msg_new:nnn { Unknown-key-for-RulesBis }
10719 {
10720   Unknown-key.\
10721   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\
10722   \c_@@_available_keys_str
10723 }
10724 {
10725   The~available~keys~are~(in~alphabetic~order):~
10726   color,~
10727   dotted,~
10728   multiplicity,~
10729   sep-color,~
10730   tikz,~and~total-width.
10731 }
10732
10733 \@@_msg_new:nnn { Unknown-key-for-Block }
10734 {
10735   Unknown-key. \
10736   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10737   \token_to_str:N \Block . \
10738   It~will~be~ignored. \
10739   \c_@@_available_keys_str
10740 }
10741 {
10742   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10743   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10744   opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10745   and~vlines.
10746 }
10747 \@@_msg_new:nnn { Unknown-key-for-Brace }
10748 {
10749   Unknown-key.\
10750   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~

```

```

10751 \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10752 It~will~be~ignored. \\
10753 \c_@@_available_keys_str
10754 }
10755 {
10756 The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10757 right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10758 right~shorten)~and~yshift.
10759 }
10760 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10761 {
10762 Unknown~key.\\
10763 The~key~' \l_keys_key_str '~is~unknown.\\
10764 It~will~be~ignored. \\
10765 \c_@@_available_keys_str
10766 }
10767 {
10768 The~available~keys~are~(in~alphabetic~order):~
10769 delimiters/color,~
10770 rules~(with~the~subkeys~'color'~and~'width'),~
10771 sub~matrix~(several~subkeys)~
10772 and~xdots~(several~subkeys).~
10773 The~latter~is~for~the~command~ \token_to_str:N \line .
10774 }
10775 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10776 {
10777 Unknown~key.\\
10778 The~key~' \l_keys_key_str '~is~unknown.\\
10779 It~will~be~ignored. \\
10780 \c_@@_available_keys_str
10781 }
10782 {
10783 The~available~keys~are~(in~alphabetic~order):~
10784 create~cell~nodes,~
10785 delimiters/color~and~
10786 sub~matrix~(several~subkeys).
10787 }
10788 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10789 {
10790 Unknown~key.\\
10791 The~key~' \l_keys_key_str '~is~unknown.\\
10792 That~key~will~be~ignored. \\
10793 \c_@@_available_keys_str
10794 }
10795 {
10796 The~available~keys~are~(in~alphabetic~order):~
10797 'delimiters/color',~
10798 'extra~height',~
10799 'hlines',~
10800 'hvlines',~
10801 'left~xshift',~
10802 'name',~
10803 'right~xshift',~
10804 'rules'~(with~the~subkeys~'color'~and~'width'),~
10805 'slim',~
10806 'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10807 and~'right~xshift').\\
10808 }
10809 \@@_msg_new:nnn { Unknown~key~for~notes }
10810 {
10811 Unknown~key.\\
10812 The~key~' \l_keys_key_str '~is~unknown.\\

```

```

10813     That~key~will~be~ignored. \\
10814     \c_@@_available_keys_str
10815 }
10816 {
10817     The~available~keys~are~(in~alphabetic~order):~
10818     bottomrule,~
10819     code~after,~
10820     code~before(+),~
10821     detect~duplicates,~
10822     enumitem~keys,~
10823     enumitem~keys~para,~
10824     para,~
10825     label~in~list,~
10826     label~in~tabular~and~
10827     style.
10828 }

10829 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10830 {
10831     Unknown~key.\\
10832     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10833     \token_to_str:N \RowStyle . \\
10834     That~key~will~be~ignored. \\
10835     \c_@@_available_keys_str
10836 }
10837 {
10838     The~available~keys~are~(in~alphabetic~order):~
10839     bold,~
10840     cell~space~top~limit(+),~
10841     cell~space~bottom~limit(+),~
10842     cell~space~limits(+),~
10843     color,~
10844     fill~(alias:~rowcolor),~
10845     nb~rows,~
10846     opacity~and~
10847     rounded~corners.
10848 }

10849 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10850 {
10851     Unknown~key.\\
10852     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10853     \token_to_str:N \NiceMatrixOptions . \\
10854     That~key~will~be~ignored. \\
10855     \c_@@_available_keys_str
10856 }
10857 {
10858     The~available~keys~are~(in~alphabetic~order):~
10859     &~in~blocks,~
10860     allow~duplicate~names,~
10861     ampersand~in~blocks,~
10862     caption~above,~
10863     cell~space~bottom~limit(+),~
10864     cell~space~limits(+),~
10865     cell~space~top~limit(+),~
10866     code~for~first~col(+),~
10867     code~for~first~row(+),~
10868     code~for~last~col(+),~
10869     code~for~last~row(+),~
10870     corners,~
10871     custom~key,~
10872     create~extra~nodes,~
10873     create~medium~nodes,~
10874     create~large~nodes,~
10875     custom~line,~

```

```

10876     delimiters~(several~subkeys),~
10877     end-of-row,~
10878     first-col,~
10879     first-row,~
10880     hlines,~
10881     hvlines,~
10882     hvlines-except-borders,~
10883     last-col,~
10884     last-row,~
10885     left-margin,~
10886     light-syntax,~
10887     light-syntax-expanded,~
10888     matrix/columns-type,~
10889     no-cell-nodes,~
10890     notes~(several~subkeys),~
10891     nullify-dots,~
10892     pgf-node-code,~
10893     renew-dots,~
10894     renew-matrix,~
10895     respect-arraystretch,~
10896     rounded-corners,~
10897     right-margin,~
10898     rules~(with~the~subkeys~'color'~and~'width'),~
10899     small,~
10900     sub-matrix~(several~subkeys),~
10901     vlines,~
10902     xdots~(several~subkeys).
10903 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10904 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10905 {
10906   Unknown~key.\\
10907   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10908   \{NiceArray\}. \\
10909   That~key~will~be~ignored. \\
10910   \c_@@_available_keys_str
10911 }
10912 {
10913   The~available~keys~are~(in~alphabetic~order):~
10914   &~in~blocks,~
10915   ampersand~in~blocks,~
10916   b,~
10917   baseline,~
10918   c,~
10919   cell-space-bottom-limit,~
10920   cell-space-limits,~
10921   cell-space-top-limit,~
10922   code-after,~
10923   code-for-first-col(+),~
10924   code-for-first-row(+),~
10925   code-for-last-col(+),~
10926   code-for-last-row(+),~
10927   columns-width,~
10928   corners,~
10929   create-extra-nodes,~
10930   create-medium-nodes,~
10931   create-large-nodes,~
10932   extra-left-margin,~
10933   extra-right-margin,~
10934   first-col,~
10935   first-row,~
10936   hlines,~

```

```

10937     hvlines,~
10938     hvlines-except-borders,~
10939     last-col,~
10940     last-row,~
10941     left-margin,~
10942     light-syntax,~
10943     light-syntax-expanded,~
10944     name,~
10945     no-cell-nodes,~
10946     nullify-dots,~
10947     pgf-node-code,~
10948     renew-dots,~
10949     respect-arraystretch,~
10950     right-margin,~
10951     rounded-corners,~
10952     rules~(with~the~subkeys~'color'~and~'width'),~
10953     small,~
10954     t,~
10955     vlines,~
10956     xdots/color,~
10957     xdots/shorten-start(+),~
10958     xdots/shorten-end(+),~
10959     xdots/shorten(+),~and~
10960     xdots/line-style.
10961 }

```

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`).

```

10962 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10963 {
10964   Unknown~key.\\
10965   The~key~' \l_keys_key_str '~is~unknown~for~the~
10966   \@@_full_name_env: . \\
10967   That~key~will~be~ignored. \\
10968   \c_@@_available_keys_str
10969 }
10970 {
10971   The~available~keys~are~(in~alphabetic~order):~
10972   &-in-blocks,~
10973   ampersand-in-blocks,~
10974   b,~
10975   baseline,~
10976   c,~
10977   cell-space-bottom-limit,~
10978   cell-space-limits,~
10979   cell-space-top-limit,~
10980   code-after,~
10981   code-for-first-col(+),~
10982   code-for-first-row(+),~
10983   code-for-last-col(+),~
10984   code-for-last-row(+),~
10985   columns-type,~
10986   columns-width,~
10987   corners,~
10988   create-extra-nodes,~
10989   create-medium-nodes,~
10990   create-large-nodes,~
10991   extra-left-margin,~
10992   extra-right-margin,~
10993   first-col,~
10994   first-row,~
10995   hlines,~
10996   hvlines,~

```

```

10997     hvlines-except-borders,~
10998     l,~
10999     last-col,~
11000     last-row,~
11001     left-margin,~
11002     light-syntax,~
11003     light-syntax-expanded,~
11004     name,~
11005     no-cell-nodes,~
11006     nullify-dots,~
11007     pgf-node-code,~
11008     r,~
11009     renew-dots,~
11010     respect-arraystretch,~
11011     right-margin,~
11012     rounded-corners,~
11013     rules~(with~the~subkeys~'color'~and~'width'),~
11014     small,~
11015     t,~
11016     vl原因,~
11017     xdots/color,~
11018     xdots/shorten-start(+),~
11019     xdots/shorten-end(+),~
11020     xdots/shorten(+)-and~
11021     xdots/line-style.
11022 }

11023 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11024 {
11025     Unknown~key.\\
11026     The~key~' \l_keys_key_str '-is~unknown~for~the~environment~
11027     \{NiceTabular\}. \\
11028     That~key~will~be~ignored. \\
11029     \c_@@_available_keys_str
11030 }
11031 {
11032     The~available~keys~are~(in~alphabetic~order):~
11033     &~in~blocks,~
11034     ampersand~in~blocks,~
11035     b,~
11036     baseline,~
11037     c,~
11038     caption,~
11039     cell-space-bottom-limit,~
11040     cell-space-limits,~
11041     cell-space-top-limit,~
11042     code~after,~
11043     code~for~first~col(+),~
11044     code~for~first~row(+),~
11045     code~for~last~col(+),~
11046     code~for~last~row(+),~
11047     columns~width,~
11048     corners,~
11049     custom~line,~
11050     create~extra~nodes,~
11051     create~medium~nodes,~
11052     create~large~nodes,~
11053     extra~left~margin,~
11054     extra~right~margin,~
11055     first~col,~
11056     first~row,~
11057     hlines,~
11058     hvlines,~
11059     hvlines-except-borders,~

```

```

11060 label,~
11061 last-col,~
11062 last-row,~
11063 left-margin,~
11064 light-syntax,~
11065 light-syntax-expanded,~
11066 name,~
11067 no-cell-nodes,~
11068 notes~(several~subkeys),~
11069 nullify-dots,~
11070 pgf-node-code,~
11071 renew-dots,~
11072 respect-arraystretch,~
11073 right-margin,~
11074 rounded-corners,~
11075 rules~(with~the~subkeys~'color'~and~'width'),~
11076 short-caption,~
11077 t,~
11078 tabularnote,~
11079 vlides,~
11080 xdots/color,~
11081 xdots/shorten-start(+),~
11082 xdots/shorten-end(+),~
11083 xdots/shorten(+)-and~
11084 xdots/line-style.
11085 }
11086 \@@_msg_new:nnn { Duplicate-name }
11087 {
11088   Duplicate-name.\
11089   The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
11090   the-same-environment-name-twice.~You-can-go-on,~but,~
11091   maybe,~you-will-have-incorrect-results-especially~
11092   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
11093   message-again,~use-the-key~'allow-duplicate-names'~in~
11094   ' \token_to_str:N \NiceMatrixOptions '~\
11095   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11096   { For-a-list-of-the-names-already-used,~type-H~<return>. }
11097 }
11098 {
11099   The-names~already-defined-in~this~document~are:~
11100   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11101 }
11102 \@@_msg_new:nn { caption-above~in~env }
11103 {
11104   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.\
11105   That-key-will-be-ignored.
11106 }
11107 \@@_msg_new:nn { show-cell-names }
11108 {
11109   There-is-no-key~'show-cell-names'~in~nicematrix.\
11110   You-should-use~the~command~\token_to_str:N \ShowCellNames\
11111   in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11112   \CodeAfter. \
11113   That-key-will-be-ignored.
11114 }
11115 \@@_msg_new:nn { Option-auto~for~columns-width }
11116 {
11117   Erroneous-use.\
11118   You-can't-give~the~value~'auto'~to~the~key~'columns-width'~here.~
11119   That-key-will-be-ignored.
11120 }
11121 \@@_msg_new:nn { NiceTabularX~without~X }

```



```

11122 {
11123     NiceTabularX~without~X.\\
11124     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
11125     However,~you~can~go~on.
11126 }
11127 \@@_msg_new:nn { Preamble~forgotten }
11128 {
11129     Preamble~forgotten.\\
11130     You~have~probably~forgotten~the~preamble~of~your~
11131     \@@_full_name_env: . \\
11132     This~error~is~fatal.
11133 }
11134 \@@_msg_new:nn { Invalid~col~number }
11135 {
11136     Invalid~column~number.\\
11137     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11138     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
11139 }
11140 \@@_msg_new:nn { Invalid~row~number }
11141 {
11142     Invalid~row~number.\\
11143     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11144     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
11145 }
11146 \@@_define_com:NNN p ( )
11147 \@@_define_com:NNN b [ ]
11148 \@@_define_com:NNN v | |
11149 \@@_define_com:NNN V \l \l
11150 \@@_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>	53
10	The environment <code>{NiceArrayWithDelims}</code>	57
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	79
13	The environment <code>{NiceMatrix}</code> and its variants	96
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	97
15	After the construction of the array	99
16	We draw the dotted lines	105
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	123
18	User commands available in the new environments	129
19	The command <code>\line</code> accessible in code-after	135
20	The command <code>\RowStyle</code>	137
21	Colors of cells, rows and columns	140
22	The vertical and horizontal rules	152
23	The empty corners	169
24	The environment <code>{NiceMatrixBlock}</code>	171
25	The extra nodes	173
26	The blocks	177
27	How to draw the dotted lines transparently	204
28	Automatic arrays	204
29	The redefinition of the command <code>\dotfill</code>	206
30	The command <code>\diagbox</code>	206

31	The keyword <code>\CodeAfter</code>	207
32	The delimiters in the preamble	208
33	The command <code>\SubMatrix</code>	209
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	218
35	The commands <code>HBrace</code> et <code>VBrace</code>	221
36	The command <code>TikzEveryCell</code>	224
37	The command <code>\ShowCellNames</code>	226
38	We process the options at package loading	227
39	About the package underscore	229
40	Error messages of the package	229