

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

April 17, 2025

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 traduction: `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-a-the-version-of-2023-11-01
12 }

13 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
14 \IfFormatAtLeastTF
15 { 2023-11-01 }
16 { }
17 { \msg_fatal:nn { nicematrix } { latex-too-old } }

18 \ProvideDocumentCommand { \IfPackageLoadedT } { m m }
19 { \IfPackageLoadedTF { #1 } { #2 } { } }
20
21 \ProvideDocumentCommand { \IfPackageLoadedF } { m m }
22 { \IfPackageLoadedTF { #1 } { } { #2 } }
```

*This document corresponds to the version 7.1b of `nicematrix`, at the date of 2025/03/30.

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

```
23 \RequirePackage { amsmath }
```

```
24 \RequirePackage { array }
```

In the version 2.6a of array, important modifications have been done for the Tagging Project.

```
25 \bool_const:Nn \c_@@_recent_array_bool
26 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
27 \bool_const:Nn \c_@@_testphase_table_bool
28 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

```
29 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
30 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
31 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
32 \cs_generate_variant:Nn \@@_error:nn { n e }
33 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
35 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
36 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

With Overleaf, 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).

```
37 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
38 {
39   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
40     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
41     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
42 }
```

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

```
43 \cs_new_protected:Npn \@@_error_or_warning:n
44 {
45   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
46     { \@@_warning:n }
47     { \@@_error:n }
48 }
```

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”.

```
49 \bool_new:N \g_@@_messages_for_Overleaf_bool
50 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
51 {
52   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
53   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
54 }
```

```
55 \cs_new_protected:Npn \@@_msg_redirect_name:nn
56 { \msg_redirect_name:nnn { nicematrix } }
57 \cs_new_protected:Npn \@@_gredirect_none:n #1
58 {
59   \group_begin:
60   \globaldefs = 1
61   \@@_msg_redirect_name:nn { #1 } { none }
62   \group_end:
63 }
```

```

64 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
65 {
66   \@@_error:n { #1 }
67   \@@_gredirect_none:n { #1 }
68 }
69 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
70 {
71   \@@_warning:n { #1 }
72   \@@_gredirect_none:n { #1 }
73 }

74 \@@_msg_new:nn { mdwtab-loaded }
75 {
76   The-packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
77   This~error~is~fatal.
78 }

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

```

2 Collecting options

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

Exemple :

`\@@_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`).

```

81 \cs_new_protected:Npn \@@_collect_options:n #1
82 {
83   \peek_meaning:NTF [
84     { \@@_collect_options:nw { #1 } }
85     { #1 { } }
86 }

```

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

```

87 \NewDocumentCommand \@@_collect_options:nw { m r[] }
88 { \@@_collect_options:nn { #1 } { #2 } }
89
90 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
91 {
92   \peek_meaning:NTF [
93     { \@@_collect_options:nnw { #1 } { #2 } }
94     { #1 { #2 } }
95 }
96
97 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
98 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

3 Technical definitions

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

```

99 \tl_const:Nn \c_@@_b_tl { b }
100 \tl_const:Nn \c_@@_c_tl { c }
101 \tl_const:Nn \c_@@_l_tl { l }
102 \tl_const:Nn \c_@@_r_tl { r }
103 \tl_const:Nn \c_@@_all_tl { all }
104 \tl_const:Nn \c_@@_dot_tl { . }
105 \str_const:Nn \c_@@_r_str { r }
106 \str_const:Nn \c_@@_c_str { c }
107 \str_const:Nn \c_@@_l_str { l }

```

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).

```

108 \tl_new:N \l_@@_argspec_tl

109 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
110 \cs_generate_variant:Nn \str_set:Nn { N o }
111 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
112 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
113 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
114 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
115 \cs_generate_variant:Nn \dim_min:nn { v }
116 \cs_generate_variant:Nn \dim_max:nn { v }

117 \hook_gput_code:nnn { begindocument } { . }
118 {
119   \IfPackageLoadedTF { tikz }
120   {

```

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.

```

121   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
122   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
123 }
124 {
125   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
126   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
127 }
128 }

```

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`).

```

129 \IfClassLoadedTF { revtex4-1 }
130 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
131 {
132   \IfClassLoadedTF { revtex4-2 }
133   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
134   {

```

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

```

135     \cs_if_exist:NT \rvtx@ifformat@geq
136     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
137     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
138   }
139 }

```

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.

```

140 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
141 {
142   \iow_now:Nn \@mainaux
143   {
144     \ExplSyntaxOn
145     \cs_if_free:NT \pgfsyspdfmark
146     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
147     \ExplSyntaxOff
148   }
149   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
150 }

```

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.

```

151 \ProvideDocumentCommand \iddots { }
152 {
153   \mathinner
154   {
155     \mkern 1 mu
156     \box_move_up:nn { 1 pt } { \hbox { . } }
157     \mkern 2 mu
158     \box_move_up:nn { 4 pt } { \hbox { . } }
159     \mkern 2 mu
160     \box_move_up:nn { 7 pt }
161     { \vbox:n { \kern 7 pt \hbox { . } } }
162     \mkern 1 mu
163   }
164 }

```

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.

```

165 \hook_gput_code:nnn { begindocument } { . }
166 {
167   \IfPackageLoadedT { booktabs }
168   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
169 }
170 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
171 {
172   \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`).

```

173   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
174   {

```

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

```

175     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
176     { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
177   }
178 }

```

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

```

179 \hook_gput_code:nnn { begindocument } { . }
180 {
181   \cs_set_protected:Npe \@@_everycr:
182   {
183     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
184     { \noalign { \@@_in_everycr: } }
185   }
186   \IfPackageLoadedTF { colortbl }
187   {
188     \cs_set_eq:NN \@@_old_cellcolor: \cellcolor
189     \cs_set_eq:NN \@@_old_rowcolor: \rowcolor
190     \cs_new_protected:Npn \@@_revert_colortbl:
191     {
192       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
193       {
194         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
195         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
196       }
197     }

```

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).

```

198     \cs_new_protected:Npn \@@_replace_columncolor:
199     {
200       \tl_replace_all:Nnn \g_@@_array_preamble_tl
201       { \columncolor }
202       { \@@_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.

```

203   }
204 }
205 {
206   \cs_new_protected:Npn \@@_revert_colortbl: { }
207   \cs_new_protected:Npn \@@_replace_columncolor:
208   { \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.

```

209   \def \CT@arc@ { }
210   \def \arrayrulecolor #1 # { \CT@arc { #1 } }
211   \def \CT@arc #1 #2
212   {
213     \dim_compare:nNt { \baselineskip } = { \c_zero_dim } { \noalign }
214     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
215   }

```

Idem for `\CT@drs@`.

```

216   \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
217   \def \CT@drs #1 #2
218   {
219     \dim_compare:nNt { \baselineskip } = { \c_zero_dim } { \noalign }
220     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
221   }
222   \def \hline
223   {

```

```

224         \noalign { \ifnum 0 = ` } \fi
225         \cs_set_eq:NN \hskip \vskip
226         \cs_set_eq:NN \vrule \hrule
227         \cs_set_eq:NN \@width \@height
228         { \CT@arc@ \vline }
229         \futurelet \reserved@a
230         \@xhline
231     }
232 }
233 }

```

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.

```

234 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
235 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
236 {
237     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
238     \int_compare:nNnT { #1 } > { \c_one_int }
239     { \multispan { \int_eval:n { #1 - 1 } } & }
240     \multispan { \int_eval:n { #2 - #1 + 1 } }
241     {
242         \CT@arc@
243         \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

244     \skip_horizontal:N \c_zero_dim
245 }

```

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`.

```

246     \everycr { }
247     \cr
248     \noalign { \skip_vertical:n { - \arrayrulewidth } }
249 }

```

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.

```

250 \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`.

```

251 { \@@_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*.

```

252 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
253 \cs_generate_variant:Nn \@@_cline_i:nn { e }
254 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
255 {
256     \tl_if_empty:nTF { #3 }
257     { \@@_cline_iii:w #1|#2-#2 \q_stop }
258     { \@@_cline_ii:w #1|#2-#3 \q_stop }
259 }
260 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
261 { \@@_cline_iii:w #1|#2-#3 \q_stop }
262 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
263 {

```

¹See question 99041 on TeX StackExchange.

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).

```

264 \int_compare:nNnT { #1 } < { #2 }
265   { \multispan { \int_eval:n { #2 - #1 } } & }
266 \multispan { \int_eval:n { #3 - #2 + 1 } }
267   {
268     \CT@arc@
269     \leaders \hrule \@height \arrayrulewidth \hfill
270     \skip_horizontal:N \c_zero_dim
271   }

```

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

```

272 \peek_meaning_remove_ignore_spaces:NTF \cline
273   { & \@@_cline_i:n { \int_eval:n { #3 + 1 } } }
274   { \everycr { } \cr }
275 }

```

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

```

276 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

277 \cs_new_protected:Npn \@@_set_CTarc:n #1
278   {
279     \tl_if_blank:nF { #1 }
280     {
281       \tl_if_head_eq_meaning:nNTF { #1 } [
282         { \def \CT@arc@ { \color #1 } }
283         { \def \CT@arc@ { \color { #1 } } }
284       ]
285     }
286 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

287 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
288   {
289     \tl_if_head_eq_meaning:nNTF { #1 } [
290       { \def \CT@drsc@ { \color #1 } }
291       { \def \CT@drsc@ { \color { #1 } } }
292     ]
293 \cs_generate_variant:Nn \@@_set_CTdrsc:n { o }

```

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

```

294 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
295   {
296     \tl_if_head_eq_meaning:nNTF { #2 } [
297       { #1 #2 }
298       { #1 { #2 } }
299     ]
300 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

301 \cs_new_protected:Npn \@@_color:n #1
302   { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
303 \cs_generate_variant:Nn \@@_color:n { o }

```

```

304 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
305   {
306     \tl_set_rescan:Nno
307       #1
308     {

```



```

309         \char_set_catcode_other:N >
310         \char_set_catcode_other:N <
311     }
312     #1
313 }

```

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

```

314 \dim_new:N \l_@@_tmpc_dim
315 \dim_new:N \l_@@_tmpd_dim
316 \dim_new:N \l_@@_tmpe_dim
317 \dim_new:N \l_@@_tmpf_dim

318 \tl_new:N \l_@@_tmpc_tl
319 \tl_new:N \l_@@_tmpd_tl

320 \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.

```

321 \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).

```

322 \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.

```

323 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
324 { \int_use:N \g_@@_env_int }

```

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

```

325 \cs_new_protected:Npn \@@_qpoint:n #1
326 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

327 \bool_new:N \l_@@_tabular_bool

```

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

```

328 \bool_new:N \g_@@_delims_bool
329 \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.

```
330 \bool_new:N \l_@@_preamble_bool
331 \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.

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

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

```
333 \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.

```
334 \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).

```
335 \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.).

```
336 \dim_new:N \l_@@_col_width_dim
337 \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.

```
338 \int_new:N \g_@@_row_total_int
339 \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).

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

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

```
341 \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.

```
342 \tl_new:N \l_@@_hpos_cell_tl
343 \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`.

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

Idem for the mono-row blocks.

```
345 \dim_new:N \g_@@_blocks_ht_dim
346 \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}`).

```
347 \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`.

```
348 \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.

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

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

```
350 \bool_new:N \l_@@_notes_detect_duplicates_bool
351 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
352 \bool_new:N \l_@@_initial_open_bool
353 \bool_new:N \l_@@_final_open_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.

```
354 \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).

```
355 \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).

```
356 \tl_new:N \l_@@_rule_color_tl
```

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

```
357 \bool_new:N \g_@@_rotate_bool
```

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

```
358 \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`).

```
359 \bool_new:N \l_@@_X_bool
360 \bool_new:N \g_@@_caption_finished_bool
```

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

```
361 \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 }`).

```
362 \tl_new:N \g_@@_aux_tl
```

During the second run, if informations concerning the current environment has been found in the `aux` file, the following flag will be raised.

```
363 \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 informations about the size of the array.

```
364 \seq_new:N \g_@@_size_seq
```

```
365 \tl_new:N \g_@@_left_delim_tl
```

```
366 \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}`).

```
367 \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`).

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

For `\multicolumn`.

```
369 \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`.

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

```
371 \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 `:`.

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

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

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

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

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

```
376 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
377 {
```

```
378   \if_mode_math: \else:
```

```
379     \@@_fatal:n { Outside-math-mode }
```

```
380   \fi:
```

```
381 }
```

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 analyse of the preamble of the array.

```
382 \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”.

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

```
384 \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`).

```
385 \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*.

```

386 \tl_new:N \g_@@_com_or_env_str
387 \tl_gset:Nn \g_@@_com_or_env_str { environment }

388 \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)`.

```

389 \cs_new:Npn \@@_full_name_env:
390 {
391   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
392   { command \space \c_backslash_str \g_@@_name_env_str }
393   { environment \space \{ \g_@@_name_env_str \} }
394 }

395 \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.

```

396 \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.

```

397 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

398 \tl_new:N \g_@@_pre_code_before_tl
399 \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 splitted in two parts because we want to control the order of execution of some instructions.

```

400 \tl_new:N \g_@@_pre_code_after_tl
401 \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.

```

402 \bool_new:N \l_@@_in_code_after_bool

```

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

```

403 \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.

```

404 \int_new:N \l_@@_old_iRow_int
405 \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).

```
406 \seq_new:N \l_@@_custom_line_commands_seq
```

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

```
407 \tl_new:N \l_@@_rules_color_tl
```

The integer `\l_@@_weight_int` will be the weight of the X column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the X columns are used in the computation of the actual width of those columns as in `tabu` (now obsolete) or `tabularray`.

```
408 \int_new:N \l_@@_weight_int
409 \int_set_eq:NN \l_@@_weight_int \c_one_int
```

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

```
410 \int_new:N \g_@@_total_X_weight_int
```

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 (the width of a column of weight n will be that dimension multiplied by n). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
411 \bool_new:N \l_@@_X_columns_aux_bool
412 \dim_new:N \l_@@_X_columns_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.

```
413 \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).

```
414 \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).

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

```
416 \tl_new:N \l_@@_code_before_tl
417 \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).

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

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

```
419 \dim_new:N \l_@@_x_initial_dim
420 \dim_new:N \l_@@_y_initial_dim
421 \dim_new:N \l_@@_x_final_dim
422 \dim_new:N \l_@@_y_final_dim
```

```

423 \dim_new:N \g_@@_dp_row_zero_dim
424 \dim_new:N \g_@@_ht_row_zero_dim
425 \dim_new:N \g_@@_ht_row_one_dim
426 \dim_new:N \g_@@_dp_ante_last_row_dim
427 \dim_new:N \g_@@_ht_last_row_dim
428 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

429 \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”.

```

430 \dim_new:N \g_@@_width_last_col_dim
431 \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.

```

432 \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!).

```

433 \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.

```

434 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocs_seq` will be erased and replaced by the value of `\g_@@_future_pos_of_blocks_seq`.

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}`.

```

435 \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.

```

436 \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`).

```

437 \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`).

```

438 \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.

```
439 \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) correspondant will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
440 \seq_new:N \g_@@_multicolumn_cells_seq
441 \seq_new:N \g_@@_multicolumn_sizes_seq
```

By default, the diagonal lines will be parallelized². 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.

```
442 \int_new:N \g_@@_ddots_int
443 \int_new:N \g_@@_iddots_int
```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_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 Δ_x and Δ_y of the first `\Iddots` diagonal.

```
444 \dim_new:N \g_@@_delta_x_one_dim
445 \dim_new:N \g_@@_delta_y_one_dim
446 \dim_new:N \g_@@_delta_x_two_dim
447 \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).

```
448 \int_new:N \l_@@_row_min_int
449 \int_new:N \l_@@_row_max_int
450 \int_new:N \l_@@_col_min_int
451 \int_new:N \l_@@_col_max_int

452 \int_new:N \l_@@_initial_i_int
453 \int_new:N \l_@@_initial_j_int
454 \int_new:N \l_@@_final_i_int
455 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
456 \int_new:N \l_@@_start_int
457 \int_set_eq:NN \l_@@_start_int \c_one_int
458 \int_new:N \l_@@_end_int
459 \int_new:N \l_@@_local_start_int
460 \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.

```
461 \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).

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

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

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

```

463 \tl_new:N \l_@@_fill_tl
464 \tl_new:N \l_@@_opacity_tl
465 \tl_new:N \l_@@_draw_tl
466 \seq_new:N \l_@@_tikz_seq
467 \clist_new:N \l_@@_borders_clist
468 \dim_new:N \l_@@_rounded_corners_dim

```

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.

```

469 \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`.

```

470 \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.

```

471 \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.

```

472 \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`).

```

473 \str_new:N \l_@@_hpos_block_str
474 \str_set:Nn \l_@@_hpos_block_str { c }
475 \bool_new:N \l_@@_hpos_of_block_cap_bool
476 \bool_new:N \l_@@_p_block_bool

```

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

```

477 \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).

```

478 \str_new:N \l_@@_vpos_block_str

```

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

```

479 \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).

```

480 \bool_new:N \l_@@_vlines_block_bool
481 \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.

```

482 \int_new:N \g_@@_block_box_int

```

```

483 \dim_new:N \l_@@_submatrix_extra_height_dim
484 \dim_new:N \l_@@_submatrix_left_xshift_dim
485 \dim_new:N \l_@@_submatrix_right_xshift_dim
486 \clist_new:N \l_@@_hlines_clist
487 \clist_new:N \l_@@_vlines_clist
488 \clist_new:N \l_@@_submatrix_hlines_clist
489 \clist_new:N \l_@@_submatrix_vlines_clist

```

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

```

490 \bool_new:N \l_@@_hlines_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.

```

491 \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`).

```

492 \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.

```

493 \int_new:N \l_@@_first_row_int
494 \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.

```

495 \int_new:N \l_@@_first_col_int
496 \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).

```

497 \int_new:N \l_@@_last_row_int
498 \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”.³

```

499 \bool_new:N \l_@@_last_row_without_value_bool

```

³We 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.

Idem for `\l_@@_last_col_without_value_bool`

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

• 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 .

```
501 \int_new:N \l_@@_last_col_int
502 \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.

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

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

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

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

Some utilities

```
505 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
506 {
```

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

```
507 \def \l_tmpa_tl { #1 }
508 \def \l_tmpb_tl { #2 }
509 }
```

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.

```
510 \cs_new_protected:Npn \@@_expand_clist:N #1
511 {
512 \clist_if_in:NnF #1 { all }
513 {
514 \clist_clear:N \l_tmpa_clist
515 \clist_map_inline:Nn #1
516 {
```

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

```
517 \tl_if_in:nnTF { ##1 } { - }
518 { \@@_cut_on_hyphen:w ##1 \q_stop }
519 {
```

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

```

520         \def \l_tmpa_tl { ##1 }
521         \def \l_tmpb_tl { ##1 }
522     }
523     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
524     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
525 }
526 \tl_set_eq:NN #1 \l_tmpa_clist
527 }
528 }
```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- `\Vdots` with both extremities open (and hence also `\Vdotsfor` in an exterior column;
- 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.

```

529 \hook_gput_code:nnn { begindocument } { . }
530 {
531     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
532     \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
533     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
534 }
```

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`.⁴
 - 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 `\c_novalue_tl`).

⁴More 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.

```
535 \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`.

```
536 \int_new:N \g_@@_tabularnote_int
537 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
538 \seq_new:N \g_@@_notes_seq
539 \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.

```
540 \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.

```
541 \seq_new:N \l_@@_notes_labels_seq
542 \newcounter { nicematrix_draft }
543 \cs_new_protected:Npn \@@_notes_format:n #1
544 {
545   \setcounter { nicematrix_draft } { #1 }
546   \@@_notes_style:n { nicematrix_draft }
547 }
```

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

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

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

```
549 \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`.

```
550 \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`.

```
551 \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`).

```
552 \hook_gput_code:nnn { begindocument } { . }
553 {
554   \IfPackageLoadedTF { enumitem }
555   {
```

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.

```

556     \newlist { tabularnotes } { enumerate } { 1 }
557     \setlist [ tabularnotes ]
558     {
559         topsep = Opt ,
560         noitemsep ,
561         leftmargin = * ,
562         align = left ,
563         labelsep = Opt ,
564         label =
565             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
566     }
567     \newlist { tabularnotes* } { enumerate* } { 1 }
568     \setlist [ tabularnotes* ]
569     {
570         afterlabel = \nobreak ,
571         itemjoin = \quad ,
572         label =
573             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
574     }

```

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`.

```

575     \NewDocumentCommand \tabularnote { o m }
576     {
577         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
578         {
579             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
580             { \@@_error:n { tabularnote~forbidden } }
581             {
582                 \bool_if:NTF \l_@@_in_caption_bool
583                 \@@_tabularnote_caption:nn
584                 \@@_tabularnote:nn
585                 { #1 } { #2 }
586             }
587         }
588     }
589 }
590 {
591     \NewDocumentCommand \tabularnote { o m }
592     {
593         \@@_error_or_warning:n { enumitem~not~loaded }
594         \@@_gredirect_none:n { enumitem~not~loaded }
595     }
596 }
597 }

598 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
599 { \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 `\c_novalue_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

600 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
601 {

```

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).

```

602     \int_zero:N \l_tmpa_int
603     \bool_if:NT \l_@@_notes_detect_duplicates_bool
604     {

```

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

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

If the user have used `\tabulernote` without the optional argument, the *label* will be the special marker expressed by `\c_novalue_tl`.

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@tabulernote`.

```

605     \int_zero:N \l_tmpb_int
606     \seq_map_indexed_inline:Nn \g_@@_notes_seq
607     {
608         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
609         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
610             {
611                 \tl_if_novalue:nTF { #1 }
612                     { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
613                     { \int_set:Nn \l_tmpa_int { ##1 } }
614                 \seq_map_break:
615             }
616         }
617         \int_if_zero:nF { \l_tmpa_int }
618         { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
619     }
620     \int_if_zero:nT { \l_tmpa_int }
621     {
622         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
623         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
624     }
625     \seq_put_right:Ne \l_@@_notes_labels_seq
626     {
627         \tl_if_novalue:nTF { #1 }
628         {
629             \@@_notes_format:n
630             {
631                 \int_eval:n
632                 {
633                     \int_if_zero:nTF { \l_tmpa_int }
634                     { \c@tabulernote }
635                     { \l_tmpa_int }
636                 }
637             }
638         }
639         { #1 }
640     }
641     \peek_meaning:NF \tabulernote
642     {

```

If the following token is *not* a `\tabulernote`, we have finished the sequence of successive commands `\tabulernote` 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`.

```

643     \hbox_set:Nn \l_tmpa_box
644     {

```

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

```

645         \@@_notes_label_in_tabular:n
646         {

```

```

647         \seq_use:Nnnn
648         \l_@@_notes_labels_seq { , } { , } { , }
649     }
650 }

```

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

```

651     \int_gdecr:N \c@tabularnote
652     \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.

```

653     \int_gincr:N \g_@@_tabularnote_int
654     \refstepcounter { tabularnote }
655     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
656     { \int_gincr:N \c@tabularnote }
657     \seq_clear:N \l_@@_notes_labels_seq
658     \bool_lazy_or:nnTF
659     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
660     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
661     {
662         \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).

```

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

```

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`.

```

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

```

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

```

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

```

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

```

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.

```

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

```


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!

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

726 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
727 {
728   \begin { pgfscope }
729   \pgfset
730     {
731       inner~sep = \c_zero_dim ,
732       minimum~size = \c_zero_dim

```

```

733     }
734     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
735     \pgfpointdiff { #3 } { #2 }
736     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
737     \pgfnode
738     { rectangle }
739     { center }
740     {
741         \vbox_to_ht:nn
742         { \dim_abs:n \l_tmpb_dim }
743         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
744     }
745     { #1 }
746     { }
747     \end { pgfscope }
748 }

```

7 The options

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

```

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

```

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

```

752 \bool_new:N \l_@@_caption_above_bool

```

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

```

753 \bool_new:N \l_@@_standard_cline_bool

```

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

```

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

```

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

```

756 \bool_new:N \l_@@_xdots_h_labels_bool

```

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

777 \bool_new:N \l_@@_amp_in_blocks_bool

```

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

```

778 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

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

```

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

```

781 \clist_new:N \l_@@_corners_clist

```

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

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

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

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

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

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

The following boolean corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

798 \bool_new:N \l_@@_delimiters_max_width_bool

799 \keys_define:nn { nicematrix / xdots }
800 {
801   shorten-start .code:n =
802     \hook_gput_code:nnn { begindocument } { . }
803     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
804   shorten-end .code:n =
805     \hook_gput_code:nnn { begindocument } { . }
806     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
807   shorten-start .value_required:n = true ,
808   shorten-end .value_required:n = true ,
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   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
817   horizontal-labels .default:n = true ,
818   line-style .code:n =
819     {
820       \bool_lazy_or:nnTF
821         { \cs_if_exist_p:N \tikzpicture }
822         { \str_if_eq_p:nn { #1 } { standard } }
823         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
824         { \@@_error:n { bad-option-for-line-style } }
825     } ,
826   line-style .value_required:n = true ,
827   color .tl_set:N = \l_@@_xdots_color_tl ,
828   color .value_required:n = true ,
829   radius .code:n =
830     \hook_gput_code:nnn { begindocument } { . }
831     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
832   radius .value_required:n = true ,
833   inter .code:n =
834     \hook_gput_code:nnn { begindocument } { . }
835     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
836   radius .value_required:n = true ,

```

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

```

837   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
838   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
839   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).

```

840   draw-first .code:n = \prg_do_nothing: ,
841   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
842 }

```

```

843 \keys_define:nn { nicematrix / rules }
844 {
845   color .tl_set:N = \l_@@_rules_color_tl ,
846   color .value_required:n = true ,
847   width .dim_set:N = \arrayrulewidth ,
848   width .value_required:n = true ,
849   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
850 }

```

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

```

851 \keys_define:nn { nicematrix / Global }
852 {
853   color-inside .code:n =
854     \@@_warning_gredirect_none:n { key-color-inside } ,
855   colortbl-like .code:n =
856     \@@_warning_gredirect_none:n { key-color-inside } ,
857   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
858   ampersand-in-blocks .default:n = true ,
859   &-in-blocks .meta:n = ampersand-in-blocks ,
860   no-cell-nodes .code:n =
861     \bool_set_true:N \l_@@_no_cell_nodes_bool
862     \cs_set_protected:Npn \@@_node_for_cell:
863       { \set@color \box_use_drop:N \l_@@_cell_box } ,
864   no-cell-nodes .value_forbidden:n = true ,
865   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
866   rounded-corners .default:n = 4 pt ,
867   custom-line .code:n = \@@_custom_line:n { #1 } ,
868   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
869   rules .value_required:n = true ,
870   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
871   standard-cline .default:n = true ,
872   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
873   cell-space-top-limit .value_required:n = true ,
874   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
875   cell-space-bottom-limit .value_required:n = true ,
876   cell-space-limits .meta:n =
877     {
878       cell-space-top-limit = #1 ,
879       cell-space-bottom-limit = #1 ,
880     } ,
881   cell-space-limits .value_required:n = true ,
882   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
883   light-syntax .code:n =
884     \bool_set_true:N \l_@@_light_syntax_bool
885     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
886   light-syntax .value_forbidden:n = true ,
887   light-syntax-expanded .code:n =
888     \bool_set_true:N \l_@@_light_syntax_bool
889     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
890   light-syntax-expanded .value_forbidden:n = true ,
891   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
892   end-of-row .value_required:n = true ,
893   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
894   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
895   last-row .int_set:N = \l_@@_last_row_int ,
896   last-row .default:n = -1 ,
897   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
898   code-for-first-col .value_required:n = true ,
899   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
900   code-for-last-col .value_required:n = true ,
901   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
902   code-for-first-row .value_required:n = true ,

```

```

903 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
904 code-for-last-row .value_required:n = true ,
905 hlines .clist_set:N = \l_@@_hlines_clist ,
906 vlines .clist_set:N = \l_@@_vlines_clist ,
907 hlines .default:n = all ,
908 vlines .default:n = all ,
909 vlines-in-sub-matrix .code:n =
910 {
911     \tl_if_single_token:nTF { #1 }
912     {
913         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
914         { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

915         { \cs_set_eq:cN { @@_#1 : } \@@_make_preamble_vlism:n }
916     }
917     { \@@_error:n { One-letter-allowed } }
918 },
919 vlines-in-sub-matrix .value_required:n = true ,
920 hvlines .code:n =
921 {
922     \bool_set_true:N \l_@@_hvlines_bool
923     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
924     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
925 },
926 hvlines-except-borders .code:n =
927 {
928     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
929     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
930     \bool_set_true:N \l_@@_hvlines_bool
931     \bool_set_true:N \l_@@_except_borders_bool
932 },
933 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.

```

934 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
935 renew-dots .value_forbidden:n = true ,
936 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
937 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
938 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
939 create-extra-nodes .meta:n =
940 { create-medium-nodes , create-large-nodes } ,
941 left-margin .dim_set:N = \l_@@_left_margin_dim ,
942 left-margin .default:n = \arraycolsep ,
943 right-margin .dim_set:N = \l_@@_right_margin_dim ,
944 right-margin .default:n = \arraycolsep ,
945 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
946 margin .default:n = \arraycolsep ,
947 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
948 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
949 extra-margin .meta:n =
950 { extra-left-margin = #1 , extra-right-margin = #1 } ,
951 extra-margin .value_required:n = true ,
952 respect-arraystretch .code:n =
953     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
954 respect-arraystretch .value_forbidden:n = true ,
955 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
956 pgf-node-code .value_required:n = true
957 }

```

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

```

958 \keys_define:nn { nicematrix / environments }
959 {
960   corners .clist_set:N = \l_@@_corners_clist ,
961   corners .default:n = { NW , SW , NE , SE } ,
962   code-before .code:n =
963   {
964     \tl_if_empty:nF { #1 }
965     {
966       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
967       \bool_set_true:N \l_@@_code_before_bool
968     }
969   } ,
970   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}`.

```

971   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
972   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
973   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
974   baseline .tl_set:N = \l_@@_baseline_tl ,
975   baseline .value_required:n = true ,
976   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)`.

```

977   \str_if_eq:eeTF { #1 } { auto }
978   { \bool_set_true:N \l_@@_auto_columns_width_bool }
979   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
980   columns-width .value_required:n = true ,
981   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.

```

982   \legacy_if:nF { measuring@ }
983   {
984     \str_set:Ne \l_@@_name_str { #1 }
985     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
986     { \@@_error:nn { Duplicate-name } { #1 } }
987     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
988   } ,
989   name .value_required:n = true ,
990   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
991   code-after .value_required:n = true ,
992 }
993 \keys_define:nn { nicematrix / notes }
994 {
995   para .bool_set:N = \l_@@_notes_para_bool ,
996   para .default:n = true ,
997   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
998   code-before .value_required:n = true ,
999   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1000   code-after .value_required:n = true ,
1001   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1002   bottomrule .default:n = true ,
1003   style .cs_set:Np = \@@_notes_style:n #1 ,
1004   style .value_required:n = true ,
1005   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1006   label-in-tabular .value_required:n = true ,
1007   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1008   label-in-list .value_required:n = true ,
1009   enumitem-keys .code:n =
1010   {
1011     \hook_gput_code:nnn { begindocument } { . }

```



```

1012     {
1013         \IfPackageLoadedT { enumitem }
1014         { \setlist* [ tabularnotes ] { #1 } }
1015     }
1016 },
1017 enumitem-keys .value_required:n = true ,
1018 enumitem-keys-para .code:n =
1019 {
1020     \hook_gput_code:nnn { begindocument } { . }
1021     {
1022         \IfPackageLoadedT { enumitem }
1023         { \setlist* [ tabularnotes* ] { #1 } }
1024     }
1025 },
1026 enumitem-keys-para .value_required:n = true ,
1027 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1028 detect-duplicates .default:n = true ,
1029 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1030 }
1031 \keys_define:nn { nicematrix / delimiters }
1032 {
1033     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1034     max-width .default:n = true ,
1035     color .tl_set:N = \l_@@_delimiters_color_tl ,
1036     color .value_required:n = true ,
1037 }

```

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

```

1038 \keys_define:nn { nicematrix }
1039 {
1040     NiceMatrixOptions .inherit:n =
1041     { nicematrix / Global } ,
1042     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1043     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1044     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1045     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1046     SubMatrix / rules .inherit:n = nicematrix / rules ,
1047     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1048     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1049     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1050     NiceMatrix .inherit:n =
1051     {
1052         nicematrix / Global ,
1053         nicematrix / environments ,
1054     } ,
1055     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1056     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1057     NiceTabular .inherit:n =
1058     {
1059         nicematrix / Global ,
1060         nicematrix / environments
1061     } ,
1062     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1063     NiceTabular / rules .inherit:n = nicematrix / rules ,
1064     NiceTabular / notes .inherit:n = nicematrix / notes ,
1065     NiceArray .inherit:n =
1066     {
1067         nicematrix / Global ,
1068         nicematrix / environments ,
1069     } ,
1070     NiceArray / xdots .inherit:n = nicematrix / xdots ,

```

```

1071   NiceArray / rules .inherit:n = nicematrix / rules ,
1072   pNiceArray .inherit:n =
1073   {
1074       nicematrix / Global ,
1075       nicematrix / environments ,
1076   } ,
1077   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1078   pNiceArray / rules .inherit:n = nicematrix / rules ,
1079 }

```

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

```

1080 \keys_define:nn { nicematrix / NiceMatrixOptions }
1081 {
1082     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1083     delimiters / color .value_required:n = true ,
1084     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1085     delimiters / max-width .default:n = true ,
1086     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1087     delimiters .value_required:n = true ,
1088     width .dim_set:N = \l_@@_width_dim ,
1089     width .value_required:n = true ,
1090     last-col .code:n =
1091     \tl_if_empty:nF { #1 }
1092     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1093     \int_zero:N \l_@@_last_col_int ,
1094     small .bool_set:N = \l_@@_small_bool ,
1095     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.

```

1096     renew-matrix .code:n = \@@_renew_matrix: ,
1097     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}`.

```

1098     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.

```

1099     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)`.

```

1100     \str_if_eq:eeTF { #1 } { auto }
1101     { \@@_error:n { Option~auto~for~columns-width } }
1102     { \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.

```

1103     allow-duplicate-names .code:n =
1104     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1105     allow-duplicate-names .value_forbidden:n = true ,
1106     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1107     notes .value_required:n = true ,
1108     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1109     sub-matrix .value_required:n = true ,
1110     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1111     matrix / columns-type .value_required:n = true ,
1112     caption-above .bool_set:N = \l_@@_caption_above_bool ,

```

```

1113     caption-above .default:n = true ,
1114     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1115 }

```

`\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.

```

1116 \NewDocumentCommand \NiceMatrixOptions { m }
1117 { \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.

```

1118 \keys_define:nn { nicematrix / NiceMatrix }
1119 {
1120     last-col .code:n = \tl_if_empty:nTF { #1 }
1121     {
1122         \bool_set_true:N \l_@@_last_col_without_value_bool
1123         \int_set:Nn \l_@@_last_col_int { -1 }
1124     }
1125     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1126     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1127     columns-type .value_required:n = true ,
1128     l .meta:n = { columns-type = l } ,
1129     r .meta:n = { columns-type = r } ,
1130     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1131     delimiters / color .value_required:n = true ,
1132     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1133     delimiters / max-width .default:n = true ,
1134     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1135     delimiters .value_required:n = true ,
1136     small .bool_set:N = \l_@@_small_bool ,
1137     small .value_forbidden:n = true ,
1138     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1139 }

```

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

```

1140 \keys_define:nn { nicematrix / NiceArray }
1141 {

```

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.

```

1142     small .bool_set:N = \l_@@_small_bool ,
1143     small .value_forbidden:n = true ,
1144     last-col .code:n = \tl_if_empty:nF { #1 }
1145     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1146     \int_zero:N \l_@@_last_col_int ,
1147     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1148     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1149     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1150 }
1151 \keys_define:nn { nicematrix / pNiceArray }
1152 {
1153     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1154     last-col .code:n = \tl_if_empty:nF { #1 }
1155     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1156     \int_zero:N \l_@@_last_col_int ,
1157     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1158     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1159     delimiters / color .value_required:n = true ,
1160     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1161     delimiters / max-width .default:n = true ,

```

```

1162     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1163     delimiters .value_required:n = true ,
1164     small .bool_set:N = \l_@@_small_bool ,
1165     small .value_forbidden:n = true ,
1166     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1167     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1168     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1169 }

```

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

```

1170 \keys_define:nn { nicematrix / NiceTabular }
1171 {

```

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.

```

1172     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1173     \bool_set_true:N \l_@@_width_used_bool ,
1174     width .value_required:n = true ,
1175     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1176     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1177     tabularnote .value_required:n = true ,
1178     caption .tl_set:N = \l_@@_caption_tl ,
1179     caption .value_required:n = true ,
1180     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1181     short-caption .value_required:n = true ,
1182     label .tl_set:N = \l_@@_label_tl ,
1183     label .value_required:n = true ,
1184     last-col .code:n = \tl_if_empty:nF { #1 }
1185     { \@@_error:n { last-col-non-empty-for~NiceArray } }
1186     \int_zero:N \l_@@_last_col_int ,
1187     r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1188     l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1189     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1190 }

```

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
```

```

1191 \keys_define:nn { nicematrix / CodeAfter }
1192 {
1193     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1194     delimiters / color .value_required:n = true ,
1195     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1196     rules .value_required:n = true ,
1197     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1198     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1199     sub-matrix .value_required:n = true ,
1200     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1201 }

```

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}`).

```
1202 \cs_new_protected:Npn \@@_cell_begin:
1203 {
```

`\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.

```
1204 \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).

```
1205 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1206 \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.

```
1207 \int_compare:nNnT { \c@jCol } = { \c_one_int }
1208 {
1209   \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1210   { \@@_begin_of_row: }
1211 }
```

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:`.

```
1212 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1213 \@@_tuning_not_tabular_begin:
1214 \@@_tuning_first_row:
1215 \@@_tuning_last_row:
1216 \g_@@_row_style_tl
1217 }
```

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.

```
1218 \cs_new_protected:Npn \@@_tuning_first_row:
1219 {
1220   \if_int_compare:w \c@iRow = \c_zero_int
1221     \if_int_compare:w \c@jCol > \c_zero_int
1222       \l_@@_code_for_first_row_tl
1223       \xglobal \colorlet { nicematrix-first-row } { . }
1224   \fi:
1225   \fi:
1226 }
```

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

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

We will use a version a little more efficient.

```
1227 \cs_new_protected:Npn \@@_tuning_last_row:
1228 {
1229   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1230     \l_@@_code_for_last_row_tl
1231     \xglobal \colorlet { nicematrix-last-row } { . }
1232   \fi:
1233 }
```

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

```
1234 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1235 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1236 {
1237   \m@th
1238   \c_math_toggle_token
```

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

```
1239   \@@_tuning_key_small:
1240 }
1241 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token
```

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.

```
1242 \cs_new_protected:Npn \@@_begin_of_row:
1243 {
1244   \int_gincr:N \c@iRow
1245   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1246   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1247   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1248   \pgfpicture
1249   \pgfrememberpicturepositiononpagetrue
1250   \pgfcoordinate
1251     { \@@_env: - row - \int_use:N \c@iRow - base }
1252     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1253   \str_if_empty:NF \l_@@_name_str
1254     {
1255       \pgfnodealias
1256         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1257         { \@@_env: - row - \int_use:N \c@iRow - base }
1258     }
1259   \endpgfpicture
1260 }
```

Remark: If the key `recreate-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 informations 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.

```

1261 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1262 {
1263   \int_if_zero:nTF { \c@iRow }
1264   {
1265     \dim_compare:nNnT
1266       { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1267       { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1268     \dim_compare:nNnT
1269       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1270       { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1271   }
1272   {
1273     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1274     {
1275       \dim_compare:nNnT
1276         { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1277         { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1278     }
1279   }
1280 }
1281 \cs_new_protected:Npn \@@_rotate_cell_box:
1282 {
1283   \box_rotate:Nn \l_@@_cell_box { 90 }
1284   \bool_if:NTF \g_@@_rotate_c_bool
1285   {
1286     \hbox_set:Nn \l_@@_cell_box
1287     {
1288       \m@th
1289       \c_math_toggle_token
1290       \vcenter { \box_use:N \l_@@_cell_box }
1291       \c_math_toggle_token
1292     }
1293   }
1294   {
1295     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1296     {
1297       \vbox_set_top:Nn \l_@@_cell_box
1298       {
1299         \vbox_to_zero:n { }
1300         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1301         \box_use:N \l_@@_cell_box
1302       }
1303     }
1304   }
1305   \bool_gset_false:N \g_@@_rotate_bool
1306   \bool_gset_false:N \g_@@_rotate_c_bool
1307 }
1308 \cs_new_protected:Npn \@@_adjust_size_box:
1309 {
1310   \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1311   {
1312     \box_set_wd:Nn \l_@@_cell_box
1313     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1314     \dim_gzero:N \g_@@_blocks_wd_dim
1315   }
1316   \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1317   {
1318     \box_set_dp:Nn \l_@@_cell_box
1319     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1320     \dim_gzero:N \g_@@_blocks_dp_dim
1321   }
1322   \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1323   {

```

```

1324     \box_set_ht:Nn \l_@@_cell_box
1325     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1326     \dim_gzero:N \g_@@_blocks_ht_dim
1327   }
1328 }
1329 \cs_new_protected:Npn \@@_cell_end:
1330 {

```

The following command is nullified in the tabulars.

```

1331     \@@_tuning_not_tabular_end:
1332     \hbox_set_end:
1333     \@@_cell_end_i:
1334   }
1335 \cs_new_protected:Npn \@@_cell_end_i:
1336 {

```

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.

```

1337     \g_@@_cell_after_hook_tl
1338     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1339     \@@_adjust_size_box:
1340     \box_set_ht:Nn \l_@@_cell_box
1341     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1342     \box_set_dp:Nn \l_@@_cell_box
1343     { \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”).

```

1344     \@@_update_max_cell_width:

```

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

```

1345     \@@_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 technic:

- 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.

```

1346     \bool_if:NTF \g_@@_empty_cell_bool
1347     { \box_use_drop:N \l_@@_cell_box }
1348     {
1349         \bool_if:NTF \g_@@_not_empty_cell_bool
1350         { \@@_print_node_cell: }
1351         {
1352             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1353             { \@@_print_node_cell: }
1354             { \box_use_drop:N \l_@@_cell_box }

```



```

1355     }
1356   }
1357   \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
1358     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1359   \bool_gset_false:N \g_@@_empty_cell_bool
1360   \bool_gset_false:N \g_@@_not_empty_cell_bool
1361 }

```

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

```

1362 \cs_new_protected:Npn \@@_update_max_cell_width:
1363 {
1364   \dim_gset:Nn \g_@@_max_cell_width_dim
1365     { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1366 }

```

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`).

```

1367 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1368 {
1369   \@@_math_toggle:
1370   \hbox_set_end:
1371   \bool_if:NF \g_@@_rotate_bool
1372     {
1373       \hbox_set:Nn \l_@@_cell_box
1374       {
1375         \makebox [ \l_@@_col_width_dim ] [ s ]
1376           { \hbox_unpack_drop:N \l_@@_cell_box }
1377       }
1378     }
1379   \@@_cell_end_i:
1380 }

```

```

1381 \pgfset
1382 {
1383   nicematrix / cell-node /.style =
1384   {
1385     inner-sep = \c_zero_dim ,
1386     minimum-width = \c_zero_dim
1387   }
1388 }

```

In the cells of a column of type `S` (of `siunitx`), we have to wrap the command `\@@_node_for_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.

```

1389 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1390 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1391 {
1392   \use:c
1393   {
1394     __siunitx_table_align_
1395     \bool_if:NTF \l__siunitx_table_text_bool
1396       { \l__siunitx_table_align_text_tl }
1397       { \l__siunitx_table_align_number_tl }
1398     :n
1399   }
1400   { #1 }
1401 }
1402 \cs_new_protected:Npn \@@_print_node_cell:
1403 { \socket_use:nn { nicematrix / siunitx-wrap } { \@@_node_for_cell: } }

```

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

```

1404 \cs_new_protected:Npn \@@_node_for_cell:
1405 {
1406   \pgfpicture
1407   \pgfsetbaseline \c_zero_dim
1408   \pgfrememberpicturepositiononpagetrue
1409   \pgfset { nicematrix / cell-node }
1410   \pgfnode
1411     { rectangle }
1412     { base }
1413     {

```

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

```

1414       \set@color
1415       \box_use_drop:N \l_@@_cell_box
1416     }
1417     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1418     { \l_@@_pgf_node_code_tl }
1419   \str_if_empty:NF \l_@@_name_str
1420   {
1421     \pgfnodealias
1422       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1423       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1424   }
1425   \endpgfpicture
1426 }

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

1427 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1428 {
1429   \cs_new_protected:Npn \@@_patch_node_for_cell:
1430   {
1431     \hbox_set:Nn \l_@@_cell_box
1432     {
1433       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1434       \hbox_overlap_left:n
1435       {
1436         \pgfsys@markposition
1437         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

In the `#1`, we will put an adjustment which is needed when the compilation is done with XeLaTeX or with the classical way `latex`, `dvips`, `ps2pdf` or Adobe Distiller (I don't know why this adjustment is mandatory...). See the use of that command `\@@_patch_node_for_cell:n` in a `\AtBeginDocument` just below.

```

1438       #1
1439     }
1440     \box_use:N \l_@@_cell_box
1441     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1442     \hbox_overlap_left:n
1443     {
1444       \pgfsys@markposition
1445       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1446     }
1447     #1
1448   }
1449 }
1450 }

```

We have no explanation for the different behaviour between the TeX engines... We put the following instructions in a `\AtBeginDocument` because you use `\sys_if_output_div_p:` and that test is available only when a backend is loaded (and we don't want to force the loading of a backend with `\sys_ensure_backend:`).

```

1451 \AtBeginDocument
1452 {
1453   \bool_lazy_or:nnTF { \sys_if_engine_xetex_p: } { \sys_if_output_dvi_p: }
1454   {
1455     \@@_patch_node_for_cell:n
1456     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1457   }
1458   { \@@_patch_node_for_cell:n { } }
1459 }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1460 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1461 {
1462   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1463   { g_@@_ #2 _ lines _ tl }
1464   {
1465     \use:c { @@ _ draw _ #2 : nnn }
1466     { \int_use:N \c@iRow }
1467     { \int_use:N \c@jCol }
1468     { \exp_not:n { #3 } }
1469   }
1470 }

1471 \cs_new_protected:Npn \@@_array:n
1472 {
1473   % \begin{macrocode}
1474   \dim_set:Nn \col@sep
1475   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1476   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1477   { \def \@halignto { } }
1478   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1479   \@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)`.

```

1480 [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1481 }
1482 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), `array` uses `\ar@ialign` instead of `\ialign`. In that case, of course, you do a saving of `\ar@ialign`.

```

1483 \bool_if:nTF
1484 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }

```

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`.

```

1485 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1486 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1487 \cs_new_protected:Npn \@@_create_row_node:
1488 {
1489   \int_compare:nNtT { \c@iRow } > { \g_@@_last_row_node_int }
1490   {
1491     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1492     \@@_create_row_node_i:
1493   }
1494 }
1495 \cs_new_protected:Npn \@@_create_row_node_i:
1496 {

```

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

```

1497   \hbox
1498   {
1499     \bool_if:NT \l_@@_code_before_bool
1500     {
1501       \vtop
1502       {
1503         \skip_vertical:N 0.5\arrayrulewidth
1504         \pgfsys@markposition
1505         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1506         \skip_vertical:N -0.5\arrayrulewidth
1507       }
1508     }
1509     \pgfpicture
1510     \pgfrememberpicturepositiononpagetrue
1511     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1512     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1513     \str_if_empty:NF \l_@@_name_str
1514     {
1515       \pgfnodealias
1516       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1517       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1518     }
1519     \endpgfpicture
1520   }
1521 }

```

```

1522 \cs_new_protected:Npn \@@_in_everycr:
1523 {
1524   \bool_if:NT \c_@@_recent_array_bool
1525   {
1526     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1527     \tbl_update_cell_data_for_next_row:
1528   }
1529   \int_gzero:N \c@jCol

```

```

1530 \bool_gset_false:N \g_@@_after_col_zero_bool
1531 \bool_if:NF \g_@@_row_of_col_done_bool
1532 {
1533   \@@_create_row_node:

```

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

```

1534   \clist_if_empty:NF \l_@@_hlines_clist
1535   {
1536     \str_if_eq:eeF \l_@@_hlines_clist { all }
1537     {
1538       \clist_if_in:NeT
1539       \l_@@_hlines_clist
1540       { \int_eval:n { \c@iRow + 1 } }
1541     }
1542   {

```

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.

```

1543     \int_compare:nNnT { \c@iRow } > { -1 }
1544     {
1545       \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1546       { \hrule height \arrayrulewidth width \c_zero_dim }
1547     }
1548   }
1549 }
1550 }
1551 }

```

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

```

1552 \cs_set_protected:Npn \@@_renew_dots:
1553 {
1554   \cs_set_eq:NN \ldots \@@_Ldots:
1555   \cs_set_eq:NN \cdots \@@_Cdots:
1556   \cs_set_eq:NN \vdots \@@_Vdots:
1557   \cs_set_eq:NN \ddots \@@_Ddots:
1558   \cs_set_eq:NN \iddots \@@_Iddots:
1559   \cs_set_eq:NN \dots \@@_Ldots:
1560   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1561 }

```

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 ⁵.

```

1562 \hook_gput_code:nnn { begindocument } { . }
1563 {
1564   \IfPackageLoadedTF { booktabs }
1565   {
1566     \cs_new_protected:Npn \@@_patch_booktabs:
1567     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1568   }
1569   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1570 }

```

⁵cf. `\nicematrix@redefine@check@rerun`

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`⁶ 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`.

```

1571 \cs_new_protected:Npn \@_some_initialization:
1572 {
1573   \@_everycr:
1574   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1575   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1576   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1577   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1578   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1579   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1580 }

```

```

1581 \cs_new_protected:Npn \@_pre_array_ii:
1582 {

```

The number of letters `X` in the preamble of the array.

```

1583   \int_gzero:N \g_@@_total_X_weight_int
1584   \@_expand_clist:N \l_@@_hlines_clist
1585   \@_expand_clist:N \l_@@_vlines_clist
1586   \@_patch_booktabs:
1587   \box_clear_new:N \l_@@_cell_box
1588   \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}`).

```

1589   \bool_if:NT \l_@@_small_bool
1590   {
1591     \def \arraystretch { 0.47 }
1592     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1593     \cs_set_eq:NN \@_tuning_key_small: \scriptstyle
1594   }

1595   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1596   {
1597     \tl_put_right:Nn \@_begin_of_row:
1598     {
1599       \pgfsys@markposition
1600       { \@_env: - row - \int_use:N \c@iRow - base }
1601     }
1602   }

```

The environment `{array}` (since version 2.6) uses internally the command `\ar@ialign` (and previously, it was `\ialign`). We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1603   \bool_if:nTF
1604   { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }

```

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

```

1605 {
1606   \def \ar@ialign
1607   {
1608     \bool_if:NT \c_@@_testphase_table_bool
1609     \tbl_init_cell_data_for_table:
1610     \@@_some_initialization:
1611     \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`.

```

1612     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1613     \halign
1614   }
1615 }

```

The following part should be deleted when we will delete the boolean `\c_@@_recent_array_bool` (when we consider the version 2.6a of `array` is required). Moreover, `revtex4-2` modifies `array` and provides commands which are meant to be the standard version of `array` but, at the date of november 2024, these commands corresponds to the *old* version of `array`, that is to say without the `\ar@ialign`.

```

1616 {
1617   \def \ialign
1618   {
1619     \@@_some_initialization:
1620     \dim_zero:N \tabskip
1621     \cs_set_eq:NN \ialign \@@_old_ialign:
1622     \halign
1623   }
1624 }

```

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).

```

1625   \bool_if:NT \c_@@_revtex_bool
1626   {
1627     \IfPackageLoadedT { colortbl }
1628     { \cs_set_protected:Npn \CT@setup { } }
1629   }

```

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).

```

1630   \cs_set_eq:NN \@@_old_ldots: \ldots
1631   \cs_set_eq:NN \@@_old_cdots: \cdots
1632   \cs_set_eq:NN \@@_old_vdots: \vdots
1633   \cs_set_eq:NN \@@_old_ddots: \ddots
1634   \cs_set_eq:NN \@@_old_iddots: \iddots
1635   \bool_if:NTF \l_@@_standard_cline_bool
1636   { \cs_set_eq:NN \cline \@@_standard_cline: }
1637   { \cs_set_eq:NN \cline \@@_cline: }
1638   \cs_set_eq:NN \Ldots \@@_Ldots:
1639   \cs_set_eq:NN \Cdots \@@_Cdots:
1640   \cs_set_eq:NN \Vdots \@@_Vdots:
1641   \cs_set_eq:NN \Ddots \@@_Ddots:
1642   \cs_set_eq:NN \Iddots \@@_Iddots:
1643   \cs_set_eq:NN \Hline \@@_Hline:
1644   \cs_set_eq:NN \Hspace \@@_Hspace:
1645   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1646   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1647   \cs_set_eq:NN \Block \@@_Block:
1648   \cs_set_eq:NN \rotate \@@_rotate:
1649   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n

```

```

1650 \cs_set_eq:NN \dotfill \@@_dotfill:
1651 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1652 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1653 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1654 \cs_set_eq:NN \TopRule \@@_TopRule
1655 \cs_set_eq:NN \MidRule \@@_MidRule
1656 \cs_set_eq:NN \BottomRule \@@_BottomRule
1657 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1658 \cs_set_eq:NN \Hbrace \@@_Hbrace
1659 \cs_set_eq:NN \Vbrace \@@_Vbrace
1660 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1661 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1662 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1663 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1664 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1665 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1666 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1667 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1668 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1669 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1670 \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:`.

```

1671 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1672 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1673 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1674 \@@_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!).

```

1675 \tl_if_exist:NT \l_@@_note_in_caption_tl
1676 {
1677   \tl_if_empty:NF \l_@@_note_in_caption_tl
1678   {
1679     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1680     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1681   }
1682 }

```

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).

```

1683 \seq_gclear:N \g_@@_multicolumn_cells_seq
1684 \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).

```

1685 \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 or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1686 \int_gzero_new: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.

```

1687 \int_gzero_new:N \g_@@_col_total_int

```



```

1688 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1689 \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.

```

1690 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1691 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1692 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1693 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1694 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1695 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1696 \tl_gclear:N \g_nicematrix_code_before_tl
1697 \tl_gclear:N \g_@@_pre_code_before_tl
1698 }

```

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

The command `\@@_pre_array:` will be executed after analyse of the keys of the environment.

```

1699 \cs_new_protected:Npn \@@_pre_array:
1700 {
1701   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1702   \int_gzero_new:N \c@iRow
1703   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1704   \int_gzero_new:N \c@jCol

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_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-column` (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).

```

1705   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1706   {
1707     \bool_set_true:N \l_@@_last_row_without_value_bool
1708     \bool_if:NT \g_@@_aux_found_bool
1709     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1710   }
1711   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1712   {
1713     \bool_if:NT \g_@@_aux_found_bool
1714     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1715   }

```

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”.

```

1716   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1717   {
1718     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1719     {
1720       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1721       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1722       \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1723       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1724     }
1725   }

1726   \seq_gclear:N \g_@@_cols_vlism_seq
1727   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```
1728 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }
```

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 clear that variable because it will be reconstructed during the creation of the array.

```
1729 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1730 \seq_gclear:N \g_@@_future_pos_of_blocks_seq
```

Idem for other sequences written on the `aux` file.

```
1731 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1732 \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.

```
1733 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1734 \@@_pre_array_ii:
```

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

```
1735 \box_clear_new:N \l_@@_the_array_box
```

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]`).

```
1736 \dim_zero_new:N \l_@@_left_delim_dim
1737 \dim_zero_new:N \l_@@_right_delim_dim
1738 \bool_if:NTF \g_@@_delims_bool
1739 {
```

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

```
1740 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1741 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1742 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1743 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1744 }
1745 {
1746 \dim_gset:Nn \l_@@_left_delim_dim
1747 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1748 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1749 }
```

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 `\c_math_toggle_token` also).

```
1750 \hbox_set:Nw \l_@@_the_array_box
1751 \skip_horizontal:N \l_@@_left_margin_dim
1752 \skip_horizontal:N \l_@@_extra_left_margin_dim
1753 \bool_if:NT \c_@@_recent_array_bool
1754 { \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).

```

1755 \m@th
1756 \c_math_toggle_token
1757 \bool_if:NTF \l_@@_light_syntax_bool
1758   { \use:c { @@-light-syntax } }
1759   { \use:c { @@-normal-syntax } }
1760 }

```

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

```

1761 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1762 {
1763   \tl_set:Nn \l_tmpa_tl { #1 }
1764   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1765     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1766   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1767   \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...

```

1768 \@@_pre_array:
1769 }

```

9 The `\CodeBefore`

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).

```

1770 \cs_new_protected:Npn \@@_pre_code_before:
1771 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in 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).

```

1772 \int_set:Nn \c_iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1773 \int_set:Nn \c_jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1774 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1775 \int_set:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }

```

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

```

1776 \pgfsys@markposition { \@@_env: - position }
1777 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1778 \pgfpicture
1779 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1780 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1781 {
1782   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1783   \pgfcoordinate { \@@_env: - row - ##1 }
1784     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1785 }

```

Now, the recreation of the col nodes.

```

1786 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1787 {
1788   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1789   \pgfcoordinate { \@@_env: - col - ##1 }
1790   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1791 }

```

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

```

1792 \@@_create_diag_nodes:

```

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

```

1793 \bool_if:NT \g_@@_recreate_cell_nodes_bool { \@@_recreate_cell_nodes: }
1794 \endpgfpicture

```

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

```

1795 \@@_create_blocks_nodes:
1796 \IfPackageLoadedT { tikz }
1797 {
1798   \tikzset
1799   {
1800     every-picture / .style =
1801     { overlay , name~prefix = \@@_env: - }
1802   }
1803 }
1804 \cs_set_eq:NN \cellcolor \@@_cellcolor
1805 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1806 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1807 \cs_set_eq:NN \rowcolor \@@_rowcolor
1808 \cs_set_eq:NN \rowcolors \@@_rowcolors
1809 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1810 \cs_set_eq:NN \arraycolor \@@_arraycolor
1811 \cs_set_eq:NN \columncolor \@@_columncolor
1812 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1813 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1814 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1815 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1816 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1817 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1818 }

```

```

1819 \cs_new_protected:Npn \@@_exec_code_before:
1820 {

```

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...

```

1821 \clist_map_inline:Nn \l_@@_corners_cells_clist
1822 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1823 \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`.

```

1824 \@@_add_to_colors_seq:nn { { nocolor } } { }
1825 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1826 \group_begin:

```

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

```

1827 \bool_if:NT \l_@@_tabular_bool { \c_math_toggle_token }

```

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`).

```
1828 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1829 { \@@_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`.

```
1830 \exp_last_unbraced:No \@@_CodeBefore_keys:
1831 \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.

```
1832 \@@_actually_color:
1833 \l_@@_code_before_tl
1834 \q_stop
1835 \bool_if:NT \l_@@_tabular_bool { \c_math_toggle_token }
1836 \group_end:
1837 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1838 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1839 }
```

```
1840 \keys_define:nn { nicematrix / CodeBefore }
1841 {
1842   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1843   create-cell-nodes .default:n = true ,
1844   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1845   sub-matrix .value_required:n = true ,
1846   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1847   delimiters / color .value_required:n = true ,
1848   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1849 }

1850 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1851 {
1852   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1853   \@@_CodeBefore:w
1854 }
```

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.

```
1855 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1856 {
1857   \bool_if:NT \g_@@_aux_found_bool
1858   {
1859     \@@_pre_code_before:
1860     \legacy_if:nF { measuring@ } { #1 }
1861   }
1862 }
```

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.

```
1863 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1864 {
1865   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
```

```

1866 {
1867   \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1868   \pgfcoordinate { \@@_env: - row - ##1 - base }
1869   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1870   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1871   {
1872     \cs_if_exist:cT
1873     { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1874     {
1875       \pgfsys@getposition
1876       { \@@_env: - ##1 - #####1 - NW }
1877       \@@_node_position:
1878       \pgfsys@getposition
1879       { \@@_env: - ##1 - #####1 - SE }
1880       \@@_node_position_i:
1881       \@@_pgf_rect_node:nnn
1882       { \@@_env: - ##1 - #####1 }
1883       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1884       { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1885     }
1886   }
1887 }
1888 \@@_create_extra_nodes:
1889 \@@_create_aliases_last:
1890 }

1891 \cs_new_protected:Npn \@@_create_aliases_last:
1892 {
1893   \int_step_inline:nn { \c@iRow }
1894   {
1895     \pgfnodealias
1896     { \@@_env: - ##1 - last }
1897     { \@@_env: - ##1 - \int_use:N \c@jCol }
1898   }
1899   \int_step_inline:nn { \c@jCol }
1900   {
1901     \pgfnodealias
1902     { \@@_env: - last - ##1 }
1903     { \@@_env: - \int_use:N \c@iRow - ##1 }
1904   }
1905   \pgfnodealias % added 2025-04-05
1906   { \@@_env: - last - last }
1907   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1908 }

1909 \cs_new_protected:Npn \@@_create_blocks_nodes:
1910 {
1911   \pgfpicture
1912   \pgf@relevantforpicturesizefalse
1913   \pgfrememberpicturepositiononpagetrue
1914   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1915   { \@@_create_one_block_node:nnnnn ##1 }
1916   \endpgfpicture
1917 }

```

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.⁷

```

1918 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5

```

⁷Moreover, 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).

```

1919 {
1920   \tl_if_empty:nF { #5 }
1921   {
1922     \@@_qpoint:n { col - #2 }
1923     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1924     \@@_qpoint:n { #1 }
1925     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1926     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1927     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1928     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1929     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1930     \@@_pgf_rect_node:nnnnn
1931     { \@@_env: - #5 }
1932     { \dim_use:N \l_tmpa_dim }
1933     { \dim_use:N \l_tmpb_dim }
1934     { \dim_use:N \l_@@_tmpc_dim }
1935     { \dim_use:N \l_@@_tmpd_dim }
1936   }
1937 }

1938 \cs_new_protected:Npn \@@_patch_for_revtext:
1939 {
1940   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1941   \cs_set_eq:NN \@array \@array@array
1942   \cs_set_eq:NN \@tabular \@tabular@array
1943   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1944   \cs_set_eq:NN \array \array@array
1945   \cs_set_eq:NN \endarray \endarray@array
1946   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1947   \cs_set_eq:NN \@mkpream \@mkpream@array
1948   \cs_set_eq:NN \@classx \@classx@array
1949   \cs_set_eq:NN \insert@column \insert@column@array
1950   \cs_set_eq:NN \@arraycr \@arraycr@array
1951   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1952   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1953 }

```

10 The environment {NiceArrayWithDelims}

```

1954 \NewDocumentEnvironment { NiceArrayWithDelims }
1955 { m m O { } m ! O { } t \CodeBefore }
1956 {
1957   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
1958   \@@_provide_pgfsyspdfmark:
1959   \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.

```

1960   \bgroup

1961   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1962   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1963   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1964   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1965   \int_gzero:N \g_@@_block_box_int
1966   \dim_gzero:N \g_@@_width_last_col_dim
1967   \dim_gzero:N \g_@@_width_first_col_dim

```

```

1968 \bool_gset_false:N \g_@@_row_of_col_done_bool
1969 \str_if_empty:NT \g_@@_name_env_str
1970 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1971 \bool_if:NTF \l_@@_tabular_bool
1972 { \mode_leave_vertical: }
1973 { \@@_test_if_math_mode: }
1974 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1975 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁸. 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.

```

1976 \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.

```

1977 \cs_if_exist:NT \tikz@library@external@loaded
1978 {
1979     \tikzexternaldisable
1980     \cs_if_exist:NT \ifstandalone
1981     { \tikzset { external / optimize = false } }
1982 }

```

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

```

1983 \int_gincr:N \g_@@_env_int
1984 \bool_if:NF \l_@@_block_auto_columns_width_bool
1985 { \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.

```

1986 \seq_gclear:N \g_@@_blocks_seq
1987 \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`.

```

1988 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1989 \seq_gclear:N \g_@@_pos_of_xdots_seq
1990 \tl_gclear_new:N \g_@@_code_before_tl
1991 \tl_gclear:N \g_@@_row_style_tl

```

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

```

1992 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1993 {
1994     \bool_gset_true:N \g_@@_aux_found_bool
1995     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1996 }
1997 { \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.

```

1998 \tl_gclear:N \g_@@_aux_tl
1999 \tl_if_empty:NF \g_@@_code_before_tl
2000 {
2001     \bool_set_true:N \l_@@_code_before_bool
2002     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2003 }
2004 \tl_if_empty:NF \g_@@_pre_code_before_tl
2005 { \bool_set_true:N \l_@@_code_before_bool }

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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`.

```

2006   \bool_if:NTF \g_@@_delims_bool
2007     { \keys_set:nn { nicematrix / pNiceArray } }
2008     { \keys_set:nn { nicematrix / NiceArray } }
2009     { #3 , #5 }

2010   \@@_set_CTarc:o \l_@@_rules_color_tl

```

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:`.

```

2011   \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2012 }

```

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

```

2013 {
2014   \bool_if:NTF \l_@@_light_syntax_bool
2015     { \use:c { end @@-light-syntax } }
2016     { \use:c { end @@-normal-syntax } }
2017   \c_math_toggle_token
2018   \skip_horizontal:N \l_@@_right_margin_dim
2019   \skip_horizontal:N \l_@@_extra_right_margin_dim
2020
2021   % awful workaround
2022   \int_if_zero:nT { \g_@@_col_total_int }
2023   {
2024     \dim_compare:nNtT { \l_@@_columns_width_dim } > { \c_zero_dim }
2025     {
2026       \skip_horizontal:n { - \l_@@_columns_width_dim }
2027       \bool_if:NTF \l_@@_tabular_bool
2028         { \skip_horizontal:n { - 2 \tabcolsep } }
2029         { \skip_horizontal:n { - 2 \arraycolsep } }
2030     }
2031   }
2032   \hbox_set_end:
2033   \bool_if:NT \c_@@_recent_array_bool
2034     { \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.

```

2035   \bool_if:NT \l_@@_width_used_bool
2036   {
2037     \int_if_zero:nT { \g_@@_total_X_weight_int }
2038     { \@@_error_or_warning:n { width-without-X-columns } }
2039   }

```

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. For a `X`-column of weight `n`, the width will be `\l_@@_X_columns_dim` multiplied by `n`.

```

2040   \int_if_zero:nF { \g_@@_total_X_weight_int } { \@@_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).

```

2041   \int_compare:nNtT { \l_@@_last_row_int } > { -2 }
2042   {
2043     \bool_if:NF \l_@@_last_row_without_value_bool

```

```

2044     {
2045         \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2046         {
2047             \@@_error:n { Wrong~last~row }
2048             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2049         }
2050     }
2051 }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\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”.⁹

```

2052     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2053     \bool_if:NTF \g_@@_last_col_found_bool
2054     { \int_gdecr:N \c@jCol }
2055     {
2056         \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2057         { \@@_error:n { last~col~not~used } }
2058     }

```

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

```

2059     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2060     \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2061     { \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. 91).

```

2062     \int_if_zero:nT { \l_@@_first_col_int }
2063     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2064     \bool_if:nTF { ! \g_@@_delims_bool }
2065     {
2066         \str_if_eq:eeTF \l_@@_baseline_tl { c }
2067         { \@@_use_arraybox_with_notes_c: }
2068         {
2069             \str_if_eq:eeTF \l_@@_baseline_tl { b }
2070             { \@@_use_arraybox_with_notes_b: }
2071             { \@@_use_arraybox_with_notes: }
2072         }
2073     }

```

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).

```

2074     {
2075         \int_if_zero:nTF { \l_@@_first_row_int }
2076         {
2077             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2078             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2079         }
2080         { \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”.¹⁰

```

2081     \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2082     {
2083         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2084         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2085     }
2086     { \dim_zero:N \l_tmpb_dim }

```

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

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

```

2087 \hbox_set:Nn \l_tmpa_box
2088 {
2089   \m@th
2090   \c_math_toggle_token
2091   \@@_color:o \l_@@_delimiters_color_tl
2092   \exp_after:wN \left \g_@@_left_delim_tl
2093   \vcenter
2094   {

```

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.

```

2095   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2096   \hbox
2097   {
2098     \bool_if:NTF \l_@@_tabular_bool
2099     { \skip_horizontal:n { - \tabcolsep } }
2100     { \skip_horizontal:n { - \arraycolsep } }
2101     \@@_use_arraybox_with_notes_c:
2102     \bool_if:NTF \l_@@_tabular_bool
2103     { \skip_horizontal:n { - \tabcolsep } }
2104     { \skip_horizontal:n { - \arraycolsep } }
2105   }

```

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

```

2106   \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2107 }
2108 \exp_after:wN \right \g_@@_right_delim_tl
2109 \c_math_toggle_token
2110 }

```

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.

```

2111 \bool_if:NTF \l_@@_delimiters_max_width_bool
2112 {
2113   \@@_put_box_in_flow_bis:nn
2114   { \g_@@_left_delim_tl }
2115   { \g_@@_right_delim_tl }
2116 }
2117 \@@_put_box_in_flow:
2118 }

```

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. 92).

```

2119 \bool_if:NT \g_@@_last_col_found_bool
2120 { \skip_horizontal:N \g_@@_width_last_col_dim }
2121 \bool_if:NT \l_@@_preamble_bool
2122 {
2123   \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2124   { \@@_warning_gredirect_none:n { columns-not-used } }
2125 }
2126 \@@_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.

```

2127 \egroup

```

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

```

2128 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2129 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2130 \iow_now:Ne \@mainaux
2131 {
2132   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2133   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2134   { \exp_not:o \g_@@_aux_tl }

```

```

2135     }
2136     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2137     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2138 }

```

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

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. For a `X`-column of weight n , the width will be `l_@@_X_columns_dim` multiplied by n .

```

2139 \cs_new_protected:Npn \@_compute_width_X:
2140 {
2141     \tl_gput_right:Ne \g_@@_aux_tl
2142     {
2143         \bool_set_true:N \l_@@_X_columns_aux_bool
2144         \dim_set:Nn \l_@@_X_columns_dim
2145         {
2146             \dim_compare:nNnTF
2147             {
2148                 \dim_abs:n
2149                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2150             }
2151             <
2152             { 0.001 pt }
2153             { \dim_use:N \l_@@_X_columns_dim }
2154             {
2155                 \dim_eval:n
2156                 {
2157                     ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2158                     / \int_use:N \g_@@_total_X_weight_int
2159                     + \l_@@_X_columns_dim
2160                 }
2161             }
2162         }
2163     }
2164 }

```

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` also.

```

2165 \cs_new_protected:Npn \@_transform_preamble:
2166 {
2167     \@_transform_preamble_i:
2168     \@_transform_preamble_ii:
2169 }

2170 \cs_new_protected:Npn \@_transform_preamble_i:
2171 {
2172     \int_gzero:N \c@jCol

```

The sequence `\g_@@_cols_vlsim_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 `vlsim`).

```

2173     \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.

```
2174 \bool_gset_false:N \g_tmpb_bool
```

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

```
2175 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2176 \int_zero:N \l_tmpa_int
2177 \tl_gclear:N \g_@@_array_preamble_tl
2178 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2179 {
2180   \tl_gset:Nn \g_@@_array_preamble_tl
2181     { ! { \skip_horizontal:N \arrayrulewidth } }
2182 }
2183 {
2184   \clist_if_in:NnT \l_@@_vlines_clist 1
2185   {
2186     \tl_gset:Nn \g_@@_array_preamble_tl
2187       { ! { \skip_horizontal:N \arrayrulewidth } }
2188   }
2189 }
```

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

```
2190 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2191 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2192 \@@_replace_columncolor:
2193 }
```

```
2194 \cs_new_protected:Npn \@@_transform_preamble_ii:
2195 {
```

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

```
2196 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2197 {
2198   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2199   { \bool_gset_true:N \g_@@_delims_bool }
2200 }
2201 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2202 \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).

```
2203 \int_if_zero:nTF { \l_@@_first_col_int }
2204 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2205 {
2206   \bool_if:NF \g_@@_delims_bool
2207   {
2208     \bool_if:NF \l_@@_tabular_bool
2209     {
2210       \clist_if_empty:NT \l_@@_vlines_clist
2211       {
2212         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2213         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2214       }
2215     }
2216   }
2217 }
```

```

2218 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2219 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2220 {
2221   \bool_if:NF \g_@@_delims_bool
2222   {
2223     \bool_if:NF \l_@@_tabular_bool
2224     {
2225       \clist_if_empty:NT \l_@@_vlines_clist
2226       {
2227         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2228         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2229       }
2230     }
2231   }
2232 }

```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it's not possible to do that in `{NiceTabular*}` (we control that with the value of `\l_@@_tabular_width_dim`).

```

2233 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2234 {

```

If the tagging of the tabulars is done (part of the Tagging Project), you don't activate that mechanism because it would create a dummy column of tagged empty cells.

```

2235   \bool_if:NF \c_@@_testphase_table_bool
2236   {
2237     \tl_gput_right:Nn \g_@@_array_preamble_tl
2238     { > { \@@_error_too_much_cols: } 1 }
2239   }
2240 }
2241 }

```

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.

```

2242 \cs_new_protected:Npn \@@_rec_preamble:n #1
2243 {

```

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.¹¹

```

2244   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2245   { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2246   {

```

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

```

2247   \cs_if_exist:cTF { NC @ find @ #1 }
2248   {
2249     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2250     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2251   }
2252   {
2253     \str_if_eq:nnTF { #1 } { S }
2254     { \@@_fatal:n { unknown~column~type~S } }
2255     { \@@_fatal:nn { unknown~column~type } { #1 } }
2256   }
2257 }
2258 }

```

¹¹We 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 c, l and r

```

2259 \cs_new_protected:Npn \@@_c: #1
2260 {
2261   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2262   \tl_gclear:N \g_@@_pre_cell_tl
2263   \tl_gput_right:Nn \g_@@_array_preamble_tl
2264     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2265   \int_gincr:N \c@jCol
2266   \@@_rec_preamble_after_col:n
2267 }

2268 \cs_new_protected:Npn \@@_l: #1
2269 {
2270   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2271   \tl_gclear:N \g_@@_pre_cell_tl
2272   \tl_gput_right:Nn \g_@@_array_preamble_tl
2273     {
2274       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2275       l
2276       < \@@_cell_end:
2277     }
2278   \int_gincr:N \c@jCol
2279   \@@_rec_preamble_after_col:n
2280 }

2281 \cs_new_protected:Npn \@@_r: #1
2282 {
2283   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2284   \tl_gclear:N \g_@@_pre_cell_tl
2285   \tl_gput_right:Nn \g_@@_array_preamble_tl
2286     {
2287       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2288       r
2289       < \@@_cell_end:
2290     }
2291   \int_gincr:N \c@jCol
2292   \@@_rec_preamble_after_col:n
2293 }

```

For ! and @

```

2294 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2295 {
2296   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2297   \@@_rec_preamble:n
2298 }
2299 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2300 \cs_new_protected:cpn { @@ _ | : } #1
2301 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2302   \int_incr:N \l_tmpa_int
2303   \@@_make_preamble_i_i:n
2304 }

2305 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2306 {
2307   \str_if_eq:nnTF { #1 } { | }
2308     { \use:c { @@ _ | : } | }
2309     { \@@_make_preamble_i_ii:nn { } #1 }
2310 }

```

```

2311 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2312 {
2313   \str_if_eq:nnTF { #2 } { [ ] }
2314   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2315   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2316 }
2317 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2318 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2319 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2320 {
2321   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2322   \tl_gput_right:Ne \g_@@_array_preamble_tl
2323   {

```

Here, the command `\dim_use:N` is mandatory.

```

2324     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2325   }
2326   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2327   {
2328     \@@_vline:n
2329     {
2330       position = \int_eval:n { \c@jCol + 1 } ,
2331       multiplicity = \int_use:N \l_tmpa_int ,
2332       total-width = \dim_use:N \l_@@_rule_width_dim ,
2333       #2
2334     }

```

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.

```

2335   }
2336   \int_zero:N \l_tmpa_int
2337   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2338   \@@_rec_preamble:n #1
2339 }

2340 \cs_new_protected:cpn { @@_> : } #1 #2
2341 {
2342   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2343   \@@_rec_preamble:n
2344 }

2345 \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.

```

2346 \keys_define:nn { nicematrix / p-column }
2347 {
2348   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2349   r .value_forbidden:n = true ,
2350   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2351   c .value_forbidden:n = true ,
2352   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2353   l .value_forbidden:n = true ,
2354   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2355   S .value_forbidden:n = true ,
2356   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2357   p .value_forbidden:n = true ,
2358   t .meta:n = p ,
2359   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2360   m .value_forbidden:n = true ,
2361   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2362   b .value_forbidden:n = true
2363 }

```


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

```

2364 \cs_new_protected:Npn \@@_p: #1
2365 {
2366   \str_set:Nn \l_@@_vpos_col_str { #1 }
Now, you look for a potential character [ after the letter of the specifier (for the options).
2367   \@@_make_preamble_ii_i:n
2368 }
2369 \cs_set_eq:NN \@@_b: \@@_p:
2370 \cs_set_eq:NN \@@_m: \@@_p:
2371 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2372 {
2373   \str_if_eq:nnTF { #1 } { [ ]
2374     { \@@_make_preamble_ii_ii:w [ ]
2375       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2376     }
2377 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2378 { \@@_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.

```

2379 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2380 {

```

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).

```

2381   \str_set:Nn \l_@@_hpos_col_str { j }
2382   \@@_keys_p_column:n { #1 }
2383   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2384 }
2385 \cs_new_protected:Npn \@@_keys_p_column:n #1
2386 { \keys_set:known { 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.

```

2387 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2388 {
2389   \use:e
2390   {
2391     \@@_make_preamble_ii_v:nnnnnnnn
2392     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2393     { \dim_eval:n { #1 } }
2394   }

```

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.

```

2395   \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2396   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2397   {

```

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

```

2398     \def \exp_not:N \l_@@_hpos_cell_tl
2399     { \str_lowercase:f { \l_@@_hpos_col_str } }
2400   }
2401   \IfPackageLoadedTF { ragged2e }
2402   {
2403     \str_case:on \l_@@_hpos_col_str
2404     {
2405       c { \Centering }
2406       l { \RaggedRight }

```

```

2407         r { \RaggedLeft }
2408     }
2409 }
2410 {
2411     \str_case:on \l_@@_hpos_col_str
2412     {
2413         c { \exp_not:N \centering }
2414         l { \exp_not:N \raggedright }
2415         r { \exp_not:N \raggedleft }
2416     }
2417 }
2418 #3
2419 }
2420 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2421 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2422 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2423 { #2 }
2424 {
2425     \str_case:onF \l_@@_hpos_col_str
2426     {
2427         { j } { c }
2428         { si } { c }
2429     }

```

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

```

2430     { \str_lowercase:f \l_@@_hpos_col_str }
2431 }
2432 }

```

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

```

2433     \int_gincr:N \c@jCol
2434     \@@_rec_preamble_after_col:n
2435 }

```

#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*.

```

2436 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2437 {
2438     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2439     {
2440         \tl_gput_right:Nn \g_@@_array_preamble_tl
2441         { > \@@_test_if_empty_for_S: }
2442     }
2443     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2444     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2445     \tl_gclear:N \g_@@_pre_cell_tl
2446     \tl_gput_right:Nn \g_@@_array_preamble_tl
2447     {
2448         > {

```

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.

```

2449         \dim_set:Nn \l_@@_col_width_dim { #2 }
2450         \bool_if:NT \c_@@_testphase_table_bool
2451           { \tag_struct_begin:n { tag = Div } }
2452         \@@_cell_begin:

```

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

```

2453         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2454         \everypar
2455         {
2456           \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2457           \everypar { }
2458         }
2459         \bool_if:NT \c_@@_testphase_table_bool { \tagpdfpara0n }

```

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

```

2460         #3

```

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

```

2461         \g_@@_row_style_tl
2462         \arraybackslash
2463         #5
2464       }
2465       #8
2466       < {
2467         #6

```

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

```

2468         \@finalstrut \@arstrutbox
2469         \use:c { end #7 }

```

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

```

2470         #4
2471         \@@_cell_end:
2472         \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2473       }
2474     }
2475   }

```

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

```

2476 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2477 {

```

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 `&`).

```

2478 \group_align_safe_begin:
2479 \peek_meaning:NTF &
2480 { \@@_the_cell_is_empty: }
2481 {
2482   \peek_meaning:NTF \\\
2483   { \@@_the_cell_is_empty: }
2484   {
2485     \peek_meaning:NTF \crcr
2486     \@@_the_cell_is_empty:
2487     \group_align_safe_end:
2488   }
2489 }
2490 }

```

```

2491 \cs_new_protected:Npn \@@_the_cell_is_empty:
2492 {
2493   \group_align_safe_end:
2494   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2495   {

```

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

```

2496     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2497     \skip_horizontal:N \l_@@_col_width_dim
2498   }
2499 }

2500 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2501 {
2502   \peek_meaning:NT \__siunitx_table_skip:n
2503   { \bool_gset_true:N \g_@@_empty_cell_bool }
2504 }

```

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.

```

2505 \cs_new_protected:Npn \@@_center_cell_box:
2506 {

```

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

```

2507   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2508   {
2509     \dim_compare:nNnT
2510     { \box_ht:N \l_@@_cell_box }
2511     >

```

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).

```

2512     { \box_ht:N \strutbox }
2513     {
2514       \hbox_set:Nn \l_@@_cell_box
2515       {
2516         \box_move_down:nn
2517         {
2518           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2519             + \baselineskip ) / 2
2520         }
2521         { \box_use:N \l_@@_cell_box }
2522       }
2523     }
2524   }
2525 }

```

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

```

2526 \cs_new_protected:Npn \@@_V: #1 #2
2527 {
2528   \str_if_eq:nnTF { #1 } { [ ] }
2529   { \@@_make_preamble_V_i:w [ ] }
2530   { \@@_make_preamble_V_i:w [ ] { #2 } }
2531 }
2532 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2533 { \@@_make_preamble_V_ii:nn { #1 } }
2534 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2

```

```

2535 {
2536   \str_set:Nn \l_@@_vpos_col_str { p }
2537   \str_set:Nn \l_@@_hpos_col_str { j }
2538   \@@_keys_p_column:n { #1 }
2539   \IfPackageLoadedTF { varwidth }
2540     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2541     {
2542       \@@_error_or_warning:n { varwidth-not-loaded }
2543       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2544     }
2545 }

```

For w and W

```

2546 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2547 \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.

```

2548 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2549 {
2550   \str_if_eq:nnTF { #3 } { s }
2551     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2552     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2553 }

```

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.

```

2554 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2555 {
2556   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2557   \tl_gclear:N \g_@@_pre_cell_tl
2558   \tl_gput_right:Nn \g_@@_array_preamble_tl
2559     {
2560       > {
2561         \dim_set:Nn \l_@@_col_width_dim { #2 }
2562         \@@_cell_begin:
2563         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2564       }
2565       c
2566       < {
2567         \@@_cell_end_for_w_s:
2568         #1
2569         \@@_adjust_size_box:
2570         \box_use_drop:N \l_@@_cell_box
2571       }
2572     }
2573   \int_gincr:N \c_jCol
2574   \@@_rec_preamble_after_col:n
2575 }

```

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

```

2576 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2577 {
2578   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2579   \tl_gclear:N \g_@@_pre_cell_tl
2580   \tl_gput_right:Nn \g_@@_array_preamble_tl
2581     {
2582       > {

```

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.

```

2583         \dim_set:Nn \l_@@_col_width_dim { #4 }
2584         \hbox_set:Nw \l_@@_cell_box
2585         \@@_cell_begin:
2586         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2587     }
2588     c
2589     < {
2590         \@@_cell_end:
2591         \hbox_set_end:
2592         #1
2593         \@@_adjust_size_box:
2594         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2595     }
2596 }

```

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

```

2597     \int_gincr:N \c@jCol
2598     \@@_rec_preamble_after_col:n
2599 }

```

```

2600 \cs_new_protected:Npn \@@_special_W:
2601 {
2602     \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2603     { \@@_warning:n { W-warning } }
2604 }

```

For `S` (of `siunitx`).

```

2605 \cs_new_protected:Npn \@@_S: #1 #2
2606 {
2607     \str_if_eq:nnTF { #2 } { [ ]
2608         { \@@_make_preamble_S:w [ ] }
2609         { \@@_make_preamble_S:w [ ] { #2 } }
2610     }
2611     \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2612     { \@@_make_preamble_S_i:n { #1 } }
2613     \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2614     {
2615         \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2616         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2617         \tl_gc_clear:N \g_@@_pre_cell_tl
2618         \tl_gput_right:Nn \g_@@_array_preamble_tl
2619         {
2620             > {

```

In the cells of a column of type `S`, we have to wrap the command `\@@_node_for_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).

```

2621         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2622         \keys_set:nn { siunitx } { #1 }
2623         \@@_cell_begin:
2624         \siunitx_cell_begin:w
2625     }
2626     c
2627     <
2628     {
2629         \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, if will stay local within the cell of the underlying `\halign`).

```

2630         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2631         {
2632             \bool_if:NTF \l__siunitx_table_text_bool
2633             { \bool_set_true:N }
2634             { \bool_set_false:N }
2635             \l__siunitx_table_text_bool
2636         }
2637     \@@_cell_end:
2638 }
2639 }

```

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

```

2640     \int_gincr:N \c@jCol
2641     \@@_rec_preamble_after_col:n
2642 }

```

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

```

2643 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2644 {
2645     \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.

```

2646     \int_if_zero:nTF { \c@jCol }
2647     {
2648         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2649         {

```

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

```

2650             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2651             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2652             \@@_rec_preamble:n #2
2653         }
2654     {
2655         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2656         \@@_make_preamble_iv:nn { #1 } { #2 }
2657     }
2658 }
2659 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2660 }
2661 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2662 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2663 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2664 {
2665     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2666     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2667     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2668     {
2669         \@@_error:nn { delimiter~after~opening } { #2 }
2670         \@@_rec_preamble:n
2671     }
2672     { \@@_rec_preamble:n #2 }
2673 }

```

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

```

2674 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2675 { \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}`).

```

2676 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2677 {
2678   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2679   \tl_if_in:nnTF { ) ] \} } { #2 }
2680   { \@@_make_preamble_v:nnn #1 #2 }
2681   {
2682     \str_if_eq:nnTF { \s_stop } { #2 }
2683     {
2684       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2685       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2686       {
2687         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2688         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2689         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2690         \@@_rec_preamble:n #2
2691       }
2692     }
2693     {
2694       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2695       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2696       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2697       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2698       \@@_rec_preamble:n #2
2699     }
2700   }
2701 }
2702 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2703 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2704 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2705 {
2706   \str_if_eq:nnTF { \s_stop } { #3 }
2707   {
2708     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2709     {
2710       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2711       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2712       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2713       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2714     }
2715     {
2716       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2717       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2718       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2719       \@@_error:nn { double~closing~delimiter } { #2 }
2720     }
2721   }
2722   {
2723     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2724     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2725     \@@_error:nn { double~closing~delimiter } { #2 }
2726     \@@_rec_preamble:n #3
2727   }
2728 }
2729 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2730 { \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 @{...}.

```

2731 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2732 {
2733   \str_if_eq:nnTF { #1 } { < }
2734   { \@@_rec_preamble_after_col_i:n }
2735   {
2736     \str_if_eq:nnTF { #1 } { @ }
2737     { \@@_rec_preamble_after_col_ii:n }
2738     {
2739       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2740       {
2741         \tl_gput_right:Nn \g_@@_array_preamble_tl
2742         { ! { \skip_horizontal:N \arrayrulewidth } }
2743       }
2744       {
2745         \clist_if_in:NcT \l_@@_vlines_clist
2746         { \int_eval:n { \c@jCol + 1 } }
2747         {
2748           \tl_gput_right:Nn \g_@@_array_preamble_tl
2749           { ! { \skip_horizontal:N \arrayrulewidth } }
2750         }
2751       }
2752       \@@_rec_preamble:n { #1 }
2753     }
2754   }
2755 }

2756 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2757 {
2758   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2759   \@@_rec_preamble_after_col:n
2760 }

```

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.

```

2761 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2762 {
2763   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2764   {
2765     \tl_gput_right:Nn \g_@@_array_preamble_tl
2766     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2767   }
2768   {
2769     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2770     {
2771       \tl_gput_right:Nn \g_@@_array_preamble_tl
2772       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2773     }
2774     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2775   }
2776   \@@_rec_preamble:n
2777 }

2778 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2779 {
2780   \tl_clear:N \l_tmpa_tl
2781   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2782   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2783 }

```

The token \NC@find is at the head of the definition of the columns type done by \newcolumnstype. We wan't that token to be no-op here.

```

2784 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2785 { \@@_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.

```

2786 \cs_new_protected:Npn \@@_X: #1 #2
2787 {
2788   \str_if_eq:nnTF { #2 } { [ ] }
2789   { \@@_make_preamble_X:w [ ] }
2790   { \@@_make_preamble_X:w [ ] #2 }
2791 }
2792 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2793 { \@@_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 as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter \l_@@_weight_int).

```

2794 \keys_define:nn { nicematrix / X-column }
2795 { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }

```

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

```

2796 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2797 {

```

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).

```

2798   \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).

```

2799   \str_set:Nn \l_@@_vpos_col_str { p }
2800   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l_tmpa_tl

```

2801   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2802   \int_compare:nNt { \l_@@_weight_int } < { \c_zero_int }
2803   {
2804     \@@_error_or_warning:n { negative-weight }
2805     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2806   }
2807   \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int

```

We test whether we know the 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).

```

2808   \bool_if:NTF \l_@@_X_columns_aux_bool
2809   {
2810     \@@_make_preamble_ii_iv:nnn
2811     { \l_@@_weight_int \l_@@_X_columns_dim }
2812     { minipage }
2813     { \@@_no_update_width: }
2814   }
2815   {
2816     \tl_gput_right:Nn \g_@@_array_preamble_tl
2817     {
2818       > {
2819         \@@_cell_begin:
2820         \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.

```

2821   \NotEmpty

```

The following code will nullify the box of the cell.

```
2822         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2823         { \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`.

```
2824         \begin { minipage } { 5 cm } \arraybackslash
2825     }
2826     c
2827     < {
2828         \end { minipage }
2829         \@@_cell_end:
2830     }
2831 }
2832 \int_gincr:N \c@jCol
2833 \@@_rec_preamble_after_col:n
2834 }
2835 }

2836 \cs_new_protected:Npn \@@_no_update_width:
2837 {
2838     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2839     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2840 }
```

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

```
2841 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2842 {
2843     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2844     { \int_eval:n { \c@jCol + 1 } }
2845     \tl_gput_right:Ne \g_@@_array_preamble_tl
2846     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2847     \@@_rec_preamble:n
2848 }
```

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.

```
2849 \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).

```
2850 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2851 { \@@_fatal:n { Preamble-forgotten } }
2852 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2853 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2854 { @@ _ \token_to_str:N \hline : }
2855 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2856 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2857 { @@ _ \token_to_str:N \hline : }
2858 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2859 { @@ _ \token_to_str:N \hline : }
2860 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2861 { @@ _ \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).

```
2862 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2863 {
```

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 that one column specifier in the preamble of `\multicolumn`.

```

2864 \multispan { #1 }
2865 \cs_set_eq:NN \@_update_max_cell_width: \prg_do_nothing:
2866 \begingroup
2867 \bool_if:NT \c_@@_testphase_table_bool
2868 { \tbl_update_multicolumn_cell_data:n { #1 } }
2869 \def \@addamp
2870 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2871 \tl_gclear:N \g_@@_preamble_tl
2872 \@_make_m_preamble:n #2 \q_stop

```

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

```

2873 \exp_args:No \mkpream \g_@@_preamble_tl
2874 \@addtopreamble \empty
2875 \endgroup
2876 \bool_if:NT \c_@@_recent_array_bool
2877 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2878 \int_compare:nNnT { #1 } > { \c_one_int }
2879 {
2880   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2881   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2882   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2883   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2884   {
2885     {
2886       \int_if_zero:nTF { \c@jCol }
2887       { \int_eval:n { \c@iRow + 1 } }
2888       { \int_use:N \c@iRow }
2889     }
2890     { \int_eval:n { \c@jCol + 1 } }
2891     {
2892       \int_if_zero:nTF { \c@jCol }
2893       { \int_eval:n { \c@iRow + 1 } }
2894       { \int_use:N \c@iRow }
2895     }
2896     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2897   { }
2898 }
2899 }

```

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

```

2900 \RenewDocumentCommand \cellcolor { 0 { } m }
2901 {
2902   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2903   {
2904     \@_rectanglecolor [ ##1 ]
2905     { \exp_not:n { ##2 } }
2906     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2907     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2908   }
2909   \ignorespaces
2910 }

```

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

```
2911 \def \@sharp { #3 }
2912 \@arstrut
2913 \@preamble
2914 \null
```

We add some lines.

```
2915 \int_gadd:Nn \c@jCol { #1 - 1 }
2916 \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
2917 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2918 \ignorespaces
2919 }
```

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`.

```
2920 \cs_new_protected:Npn \@_make_m_preamble:n #1
2921 {
2922   \str_case:nnF { #1 }
2923   {
2924     c { \@_make_m_preamble_i:n #1 }
2925     l { \@_make_m_preamble_i:n #1 }
2926     r { \@_make_m_preamble_i:n #1 }
2927     > { \@_make_m_preamble_ii:nn #1 }
2928     ! { \@_make_m_preamble_ii:nn #1 }
2929     @ { \@_make_m_preamble_ii:nn #1 }
2930     | { \@_make_m_preamble_iii:n #1 }
2931     p { \@_make_m_preamble_iv:nnn t #1 }
2932     m { \@_make_m_preamble_iv:nnn c #1 }
2933     b { \@_make_m_preamble_iv:nnn b #1 }
2934     w { \@_make_m_preamble_v:nnnn { } #1 }
2935     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2936     \q_stop { }
2937   }
2938   {
2939     \cs_if_exist:cTF { NC @ find @ #1 }
2940     {
2941       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2942       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2943     }
2944     {
2945       \str_if_eq:nnTF { #1 } { S }
2946       { \@_fatal:n { unknown~column~type~S } }
2947       { \@_fatal:nn { unknown~column~type } { #1 } }
2948     }
2949   }
2950 }
```

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

```
2951 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2952 {
2953   \tl_gput_right:Nn \g_@@_preamble_tl
2954   {
2955     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2956     #1
2957     < \@_cell_end:
2958   }
2959 }
```

We test for the presence of a `<`.

```
2959 \@_make_m_preamble_x:n
2960 }
```

For >, ! and @

```

2961 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2962 {
2963   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2964   \@@_make_m_preamble:n
2965 }

```

For |

```

2966 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2967 {
2968   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2969   \@@_make_m_preamble:n
2970 }

```

For p, m and b

```

2971 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2972 {
2973   \tl_gput_right:Nn \g_@@_preamble_tl
2974   {
2975     > {
2976       \@@_cell_begin:
2977       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2978       \mode_leave_vertical:
2979       \arraybackslash
2980       \vrule height \box_ht:N \@@_arstrutbox depth 0 pt width 0 pt
2981     }
2982     c
2983     < {
2984       \vrule height 0 pt depth \box_dp:N \@@_arstrutbox width 0 pt
2985       \end { minipage }
2986       \@@_cell_end:
2987     }
2988   }

```

We test for the presence of a <.

```

2989   \@@_make_m_preamble_x:n
2990 }

```

For w and W

```

2991 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2992 {
2993   \tl_gput_right:Nn \g_@@_preamble_tl
2994   {
2995     > {
2996       \dim_set:Nn \l_@@_col_width_dim { #4 }
2997       \hbox_set:Nw \l_@@_cell_box
2998       \@@_cell_begin:
2999       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3000     }
3001     c
3002     < {
3003       \@@_cell_end:
3004       \hbox_set_end:
3005       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3006       #1
3007       \@@_adjust_size_box:
3008       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3009     }
3010   }

```

We test for the presence of a <.

```

3011   \@@_make_m_preamble_x:n
3012 }

```

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

```

3013 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3014 {
3015   \str_if_eq:nnTF { #1 } { < }
3016     { \@@_make_m_preamble_ix:n }
3017     { \@@_make_m_preamble:n { #1 } }
3018 }
3019 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3020 {
3021   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3022   \@@_make_m_preamble_x:n
3023 }

```

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`).

```

3024 \cs_new_protected:Npn \@@_put_box_in_flow:
3025 {
3026   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3027   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3028   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3029     { \box_use_drop:N \l_tmpa_box }
3030     { \@@_put_box_in_flow_i: }
3031 }

```

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

```

3032 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3033 {
3034   \pgfpicture
3035     \@@_qpoint:n { row - 1 }
3036     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3037     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3038     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3039     \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).

```

3040   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3041   {
3042     \int_set:Nn \l_tmpa_int
3043     {
3044       \str_range:Nnn
3045         \l_@@_baseline_tl
3046         6
3047         { \tl_count:o \l_@@_baseline_tl }
3048     }
3049     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3050   }
3051   {
3052     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3053     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3054     {
3055       \str_if_eq:onTF \l_@@_baseline_tl { b }
3056       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3057       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3058     }
3059     \bool_lazy_or:nnT
3060       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3061       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }

```

```

3062         {
3063             \@@_error:n { bad-value-for-baseline }
3064             \int_set_eq:NN \l_tmpa_int \c_one_int
3065         }
3066         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3067         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3068     }
3069     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3070     \endpgfpicture
3071     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3072     \box_use_drop:N \l_tmpa_box
3073 }

```

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).

```

3074 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3075 {

```

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.

```

3076     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3077     {
3078         \int_compare:nNnT { \c_jCol } > { \c_one_int }
3079         {
3080             \box_set_wd:Nn \l_@@_the_array_box
3081             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3082         }
3083     }

```

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

```

3084     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3085     \bool_if:NT \l_@@_caption_above_bool
3086     {
3087         \tl_if_empty:NF \l_@@_caption_tl
3088         {
3089             \bool_set_false:N \g_@@_caption_finished_bool
3090             \int_gzero:N \c@tabularnote
3091             \@@_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).

```

3092         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3093         {
3094             \tl_gput_right:Ne \g_@@_aux_tl
3095             {
3096                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3097                 { \int_use:N \g_@@_notes_caption_int }
3098             }
3099             \int_gzero:N \g_@@_notes_caption_int
3100         }
3101     }
3102 }

```


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

```

3103   \hbox
3104   {
3105     \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right now 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.

```

3106     \@@_create_extra_nodes:
3107     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3108   }

```

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 if compiles several twice its tabular).

```

3109   \bool_lazy_any:nT
3110   {
3111     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3112     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3113     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3114   }
3115   \@@_insert_tabularnotes:
3116   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3117   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3118   \end { minipage }
3119 }

```

```

3120 \cs_new_protected:Npn \@@_insert_caption:
3121 {
3122   \tl_if_empty:NF \l_@@_caption_tl
3123   {
3124     \cs_if_exist:NTF \c@type
3125     { \@@_insert_caption_i: }
3126     { \@@_error:n { caption~outside~float } }
3127   }
3128 }

```

```

3129 \cs_new_protected:Npn \@@_insert_caption_i:
3130 {
3131   \group_begin:

```

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

```

3132   \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.

```

3133   \IfPackageLoadedT { floatrow }
3134   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3135   \tl_if_empty:NTF \l_@@_short_caption_tl
3136   { \caption }
3137   { \caption [ \l_@@_short_caption_tl ] }
3138   { \l_@@_caption_tl }

```

In some circonstancies (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.

```

3139 \bool_if:NF \g_@@_caption_finished_bool
3140 {
3141   \bool_gset_true:N \g_@@_caption_finished_bool
3142   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3143   \int_gzero:N \c@tabularnote
3144 }
3145 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3146 \group_end:
3147 }
3148 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3149 {
3150   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3151   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3152 }
3153 \cs_new_protected:Npn \@@_insert_tabularnotes:
3154 {
3155   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3156   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3157   \skip_vertical:N 0.65ex

```

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

```

3158 \group_begin:
3159 \l_@@_notes_code_before_tl
3160 \tl_if_empty:NF \g_@@_tabularnote_tl
3161 {
3162   \g_@@_tabularnote_tl \par
3163   \tl_gclear:N \g_@@_tabularnote_tl
3164 }

```

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.

```

3165 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3166 {
3167   \bool_if:NTF \l_@@_notes_para_bool
3168   {
3169     \begin { tabularnotes* }
3170     \seq_map_inline:Nn \g_@@_notes_seq
3171       { \@@_one_tabularnote:nn ##1 }
3172     \strut
3173     \end { tabularnotes* }

```

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

```

3174 \par
3175 }
3176 {
3177   \tabularnotes
3178   \seq_map_inline:Nn \g_@@_notes_seq
3179     { \@@_one_tabularnote:nn ##1 }
3180   \strut
3181   \endtabularnotes
3182 }
3183 }
3184 \unskip
3185 \group_end:
3186 \bool_if:NT \l_@@_notes_bottomrule_bool
3187 {
3188   \IfPackageLoadedTF { booktabs }
3189   {

```

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

```

3190 \skip_vertical:N \aboverulesep

```

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

```

3191         { \CT@arc@ \hrule height \heavyrulewidth }
3192     }
3193     { \@_error_or_warning:n { bottomrule-without-booktabs } }
3194 }
3195 \l_@@_notes_code_after_tl
3196 \seq_gclear:N \g_@@_notes_seq
3197 \seq_gclear:N \g_@@_notes_in_caption_seq
3198 \int_gzero:N \c@tabularnote
3199 }

```

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.

```

3200 \cs_set_protected:Npn \@_one_tabularnote:nn #1
3201 {
3202     \tl_if_novalue:nTF { #1 }
3203     { \item }
3204     { \item [ \@_notes_label_in_list:n { #1 } ] }
3205 }

```

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.

```

3206 \cs_new_protected:Npn \@_use_arraybox_with_notes_b:
3207 {
3208     \pgfpicture
3209     \@_qpoint:n { row - 1 }
3210     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3211     \@_qpoint:n { row - \int_use:N \c@iRow - base }
3212     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3213     \endpgfpicture
3214     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3215     \int_if_zero:nT { \l_@@_first_row_int }
3216     {
3217         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3218         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3219     }
3220     \box_move_up:nn \g_tmpa_dim { \hbox { \@_use_arraybox_with_notes_c: } }
3221 }

```

Now, the general case.

```

3222 \cs_new_protected:Npn \@_use_arraybox_with_notes:
3223 {

```

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

```

3224     \str_if_eq:eeT \l_@@_baseline_tl { t }
3225     { \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`.

```

3226     \pgfpicture
3227     \@_qpoint:n { row - 1 }
3228     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3229     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3230     {
3231         \int_set:Nn \l_tmpa_int
3232         {
3233             \str_range:Nnn
3234             \l_@@_baseline_tl
3235             { 6 }
3236             { \tl_count:o \l_@@_baseline_tl }

```

```

3237     }
3238     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3239   }
3240   {
3241     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3242     \bool_lazy_or:nnT
3243       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3244       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3245     {
3246       \@@_error:n { bad-value-for~baseline }
3247       \int_set:Nn \l_tmpa_int 1
3248     }
3249     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3250   }
3251   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3252   \endpgfpicture
3253   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3254   \int_if_zero:nT { \l_@@_first_row_int }
3255   {
3256     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3257     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3258   }
3259   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3260 }

```

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.

```

3261 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3262 {

```

We will compute the real width of both delimiters used.

```

3263   \dim_zero_new:N \l_@@_real_left_delim_dim
3264   \dim_zero_new:N \l_@@_real_right_delim_dim
3265   \hbox_set:Nn \l_tmpb_box
3266   {
3267     \m@th % added 2024/11/21
3268     \c_math_toggle_token
3269     \left #1
3270     \vcenter
3271     {
3272       \vbox_to_ht:nn
3273         { \box_ht_plus_dp:N \l_tmpa_box }
3274       { }
3275     }
3276     \right .
3277     \c_math_toggle_token
3278   }
3279   \dim_set:Nn \l_@@_real_left_delim_dim
3280   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3281   \hbox_set:Nn \l_tmpb_box
3282   {
3283     \m@th % added 2024/11/21
3284     \c_math_toggle_token
3285     \left .
3286     \vbox_to_ht:nn
3287       { \box_ht_plus_dp:N \l_tmpa_box }
3288     { }
3289     \right #2
3290     \c_math_toggle_token
3291   }
3292   \dim_set:Nn \l_@@_real_right_delim_dim
3293   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3294     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3295     \@@_put_box_in_flow:
3296     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3297 }

```

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).

```

3298 \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).

```

3299 {
3300   \peek_remove_spaces:n
3301   {
3302     \peek_meaning:NTF \end
3303     { \@@_analyze_end:Nn }
3304     {
3305       \@@_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`).

```

3306       \@@_array:o \g_@@_array_preamble_tl
3307     }
3308   }
3309 }
3310 {
3311   \@@_create_col_nodes:
3312   \endarray
3313 }

```

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

```

3314 \NewDocumentEnvironment { @@-light-syntax } { b }
3315 {

```

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`.

```

3316   \tl_if_empty:nT { #1 }
3317   { \@@_fatal:n { empty-environment } }
3318   \tl_if_in:nnT { #1 } { & }
3319   { \@@_fatal:n { ampersand-in-light-syntax } }
3320   \tl_if_in:nnT { #1 } { \ }
3321   { \@@_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`.

```

3322   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3323 }

```

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.

```

3324 {
3325   \@@_create_col_nodes:
3326   \endarray
3327 }

3328 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3329 {
3330   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

The body of the array, which is stored in the argument `#1`, is now splitted into items (and *not* tokens).

```

3331   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3332   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3333   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3334     { \seq_set_split:Nee }
3335     { \seq_set_split:Non }
3336   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3337   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3338   \tl_if_empty:NF \l_tmpa_tl
3339   { \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.

```

3340   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3341     { \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`).

```

3342   \tl_build_begin:N \l_@@_new_body_tl
3343   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3344   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3345   \@@_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).

```

3346   \seq_map_inline:Nn \l_@@_rows_seq
3347   {
3348     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3349     \@@_line_with_light_syntax:n { ##1 }
3350   }
3351   \tl_build_end:N \l_@@_new_body_tl

3352   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3353   {
3354     \int_set:Nn \l_@@_last_col_int
3355       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3356   }

```

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.

```

3357   \@@_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`).

```

3358   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3359 }

```

```

3360 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3361 {
3362   \seq_clear_new:N \l_@@_cells_seq
3363   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3364   \int_set:Nn \l_@@_nb_cols_int
3365   {
3366     \int_max:nn
3367     { \l_@@_nb_cols_int }
3368     { \seq_count:N \l_@@_cells_seq }
3369   }
3370   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3371   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3372   \seq_map_inline:Nn \l_@@_cells_seq
3373   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3374 }
3375 \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`.

```

3376 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3377 {
3378   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3379   { \@@_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.

```

3380   \end { #2 }
3381 }

```

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`).

```

3382 \cs_new:Npn \@@_create_col_nodes:
3383 {
3384   \crrr
3385   \int_if_zero:nT { \l_@@_first_col_int }
3386   {
3387     \omit
3388     \hbox_overlap_left:n
3389     {
3390       \bool_if:NT \l_@@_code_before_bool
3391       { \pgfsys@markposition { \@@_env: - col - 0 } }
3392       \pgfpicture
3393       \pgfrememberpicturepositiononpagetrue
3394       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3395       \str_if_empty:NF \l_@@_name_str
3396       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3397       \endpgfpicture
3398       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3399     }
3400     &
3401   }
3402   \omit

```

The following instruction must be put after the instruction `\omit`.

```

3403   \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`).

```

3404   \int_if_zero:nTF { \l_@@_first_col_int }
3405   {
3406     \bool_if:NT \l_@@_code_before_bool

```

```

3407     {
3408         \hbox
3409         {
3410             \skip_horizontal:n { -0.5 \arrayrulewidth }
3411             \pgfsys@markposition { \@@_env: - col - 1 }
3412             \skip_horizontal:n { 0.5 \arrayrulewidth }
3413         }
3414     }
3415     \pgfpicture
3416     \pgfrememberpicturepositiononpagetrue
3417     \pgfcoordinate { \@@_env: - col - 1 }
3418     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3419     \str_if_empty:NF \l_@@_name_str
3420     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3421     \endpgfpicture
3422 }
3423 {
3424     \bool_if:NT \l_@@_code_before_bool
3425     {
3426         \hbox
3427         {
3428             \skip_horizontal:n { 0.5 \arrayrulewidth }
3429             \pgfsys@markposition { \@@_env: - col - 1 }
3430             \skip_horizontal:n { -0.5 \arrayrulewidth }
3431         }
3432     }
3433     \pgfpicture
3434     \pgfrememberpicturepositiononpagetrue
3435     \pgfcoordinate { \@@_env: - col - 1 }
3436     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3437     \str_if_empty:NF \l_@@_name_str
3438     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3439     \endpgfpicture
3440 }

```

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.

```

3441     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3442     \bool_if:NF \l_@@_auto_columns_width_bool
3443     { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3444     {
3445         \bool_lazy_and:nnTF
3446         { \l_@@_auto_columns_width_bool }
3447         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3448         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3449         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3450         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3451     }
3452     \skip_horizontal:N \g_tmpa_skip
3453     \hbox
3454     {
3455         \bool_if:NT \l_@@_code_before_bool
3456         {
3457             \hbox
3458             {
3459                 \skip_horizontal:n { -0.5 \arrayrulewidth }
3460                 \pgfsys@markposition { \@@_env: - col - 2 }
3461                 \skip_horizontal:n { 0.5 \arrayrulewidth }
3462             }

```



```

3463     }
3464     \pgfpicture
3465     \pgfrememberpicturepositiononpagetrue
3466     \pgfcoordinate { \@@_env: - col - 2 }
3467     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3468     \str_if_empty:NF \l_@@_name_str
3469     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3470     \endpgfpicture
3471 }

```

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.

```

3472     \int_gset_eq:NN \g_tmpa_int \c_one_int
3473     \bool_if:NTF \g_@@_last_col_found_bool
3474     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3475     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3476     {
3477     &
3478     \omit
3479     \int_gincr:N \g_tmpa_int

```

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

```

3480     \skip_horizontal:N \g_tmpa_skip
3481     \bool_if:NT \l_@@_code_before_bool
3482     {
3483     \hbox
3484     {
3485     \skip_horizontal:n { -0.5 \arrayrulewidth }
3486     \pgfsys@markposition
3487     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3488     \skip_horizontal:n { 0.5 \arrayrulewidth }
3489     }
3490     }

```

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

```

3491     \pgfpicture
3492     \pgfrememberpicturepositiononpagetrue
3493     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3494     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3495     \str_if_empty:NF \l_@@_name_str
3496     {
3497     \pgfnodealias
3498     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3499     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3500     }
3501     \endpgfpicture
3502 }

3503 &
3504 \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentioned by Joao Luis Soares by mail.

```

3505     \int_if_zero:nT { \g_@@_col_total_int }
3506     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3507     \skip_horizontal:N \g_tmpa_skip
3508     \int_gincr:N \g_tmpa_int
3509     \bool_lazy_any:nF
3510     {
3511     \g_@@_delims_bool
3512     \l_@@_tabular_bool
3513     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3514     \l_@@_exterior_arraycolsep_bool
3515     \l_@@_bar_at_end_of_pream_bool

```

```

3516     }
3517     { \skip_horizontal:n { - \col@sep } }
3518 \bool_if:NT \l_@@_code_before_bool
3519     {
3520         \hbox
3521         {
3522             \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.

```

3523         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3524         { \skip_horizontal:n { - \arraycolsep } }
3525         \pgfsys@markposition
3526         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3527         \skip_horizontal:n { 0.5 \arrayrulewidth }
3528         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3529         { \skip_horizontal:N \arraycolsep }
3530     }
3531 }
3532 \pgfpicture
3533 \pgfrememberpicturepositiononpagetrue
3534 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3535 {
3536     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3537     {
3538         \pgfpoint
3539         { - 0.5 \arrayrulewidth - \arraycolsep }
3540         \c_zero_dim
3541     }
3542     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3543 }
3544 \str_if_empty:NF \l_@@_name_str
3545 {
3546     \pgfnodealias
3547     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3548     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3549 }
3550 \endpgfpicture

3551 \bool_if:NT \g_@@_last_col_found_bool
3552 {
3553     \hbox_overlap_right:n
3554     {
3555         \skip_horizontal:N \g_@@_width_last_col_dim
3556         \skip_horizontal:N \col@sep
3557         \bool_if:NT \l_@@_code_before_bool
3558         {
3559             \pgfsys@markposition
3560             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3561         }
3562         \pgfpicture
3563         \pgfrememberpicturepositiononpagetrue
3564         \pgfcoordinate
3565         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3566         \pgfpointorigin
3567         \str_if_empty:NF \l_@@_name_str
3568         {
3569             \pgfnodealias
3570             {
3571                 \l_@@_name_str - col
3572                 - \int_eval:n { \g_@@_col_total_int + 1 }
3573             }

```

```

3574         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3575     }
3576     \endpgfpicture
3577 }
3578 }
3579 % \cr
3580 }

```

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

```

3581 \tl_const:Nn \c_@@_preamble_first_col_tl
3582 {
3583     >
3584     {

```

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

```

3585         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3586         \bool_gset_true:N \g_@@_after_col_zero_bool
3587         \@@_begin_of_row:
3588         \hbox_set:Nw \l_@@_cell_box
3589         \@@_math_toggle:
3590         \@@_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”.

```

3591         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3592         {
3593             \bool_lazy_or:nnT
3594             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3595             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3596             {
3597                 \l_@@_code_for_first_col_tl
3598                 \xglobal \colorlet { nicematrix-first-col } { . }
3599             }
3600         }
3601     }

```

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`.

```

3602     l
3603     <
3604     {
3605         \@@_math_toggle:
3606         \hbox_set_end:
3607         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3608         \@@_adjust_size_box:
3609         \@@_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.

```

3610         \dim_gset:Nn \g_@@_width_first_col_dim
3611         { \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.

```

3612         \hbox_overlap_left:n
3613         {
3614             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3615             { \@@_node_for_cell: }
3616             { \box_use_drop:N \l_@@_cell_box }
3617             \skip_horizontal:N \l_@@_left_delim_dim
3618             \skip_horizontal:N \l_@@_left_margin_dim
3619             \skip_horizontal:N \l_@@_extra_left_margin_dim
3620         }
3621         \bool_gset_false:N \g_@@_empty_cell_bool

```

```

3622     \skip_horizontal:n { -2 \col@sep }
3623   }
3624 }

```

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

```

3625 \tl_const:Nn \c_@@_preamble_last_col_tl
3626 {
3627   >
3628   {
3629     \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).

```

3630     \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.

```

3631     \bool_gset_true:N \g_@@_last_col_found_bool
3632     \int_gincr:N \c@jCol
3633     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3634     \hbox_set:Nw \l_@@_cell_box
3635     \@@_math_toggle:
3636     \@@_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”.

```

3637     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3638     {
3639       \bool_lazy_or:nnT
3640       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3641       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3642       {
3643         \l_@@_code_for_last_col_tl
3644         \xglobal \colorlet { nicematrix-last-col } { . }
3645       }
3646     }
3647   }
3648   1
3649   <
3650   {
3651     \@@_math_toggle:
3652     \hbox_set_end:
3653     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3654     \@@_adjust_size_box:
3655     \@@_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.

```

3656     \dim_gset:Nn \g_@@_width_last_col_dim
3657     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3658     \skip_horizontal:n { -2 \col@sep }

```

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

```

3659     \hbox_overlap_right:n
3660     {
3661       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3662       {
3663         \skip_horizontal:N \l_@@_right_delim_dim
3664         \skip_horizontal:N \l_@@_right_margin_dim
3665         \skip_horizontal:N \l_@@_extra_right_margin_dim
3666         \@@_node_for_cell:
3667       }
3668     }
3669     \bool_gset_false:N \g_@@_empty_cell_bool
3670   }
3671 }

```

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

```

3672 \NewDocumentEnvironment { NiceArray } { }
3673 {
3674   \bool_gset_false:N \g_@@_delims_bool
3675   \str_if_empty:NT \g_@@_name_env_str
3676   { \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).

```

3677   \NiceArrayWithDelims . .
3678 }
3679 { \endNiceArrayWithDelims }

```

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

```

3680 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3681 {
3682   \NewDocumentEnvironment { #1 NiceArray } { }
3683   {
3684     \bool_gset_true:N \g_@@_delims_bool
3685     \str_if_empty:NT \g_@@_name_env_str
3686     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3687     \@@_test_if_math_mode:
3688     \NiceArrayWithDelims #2 #3
3689   }
3690   { \endNiceArrayWithDelims }
3691 }
3692 \@@_def_env:NNN p ( )
3693 \@@_def_env:NNN b [ ]
3694 \@@_def_env:NNN B \{ \}
3695 \@@_def_env:NNN v | |
3696 \@@_def_env:NNN V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3697 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3698 {
3699   \bool_set_false:N \l_@@_preamble_bool
3700   \tl_clear:N \l_tmpa_tl
3701   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3702   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3703   \tl_put_right:Nn \l_tmpa_tl
3704   {
3705     *
3706     {
3707       \int_case:nnF \l_@@_last_col_int
3708       {
3709         { -2 } { \c@MaxMatrixCols }
3710         { -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).

```

3711       }
3712       { \int_eval:n { \l_@@_last_col_int - 1 } }
3713     }
3714     { #2 }
3715   }
3716   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3717   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3718 }

```

```

3719 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3720 \clist_map_inline:nn { p , b , B , v , V }
3721 {
3722   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3723   {
3724     \bool_gset_true:N \g_@@_delims_bool
3725     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3726     \int_if_zero:nT { \l_@@_last_col_int }
3727     {
3728       \bool_set_true:N \l_@@_last_col_without_value_bool
3729       \int_set:Nn \l_@@_last_col_int { -1 }
3730     }
3731     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3732     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3733   }
3734   { \use:c { end #1 NiceArray } }
3735 }

```

We define also an environment {NiceMatrix}

```

3736 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3737 {
3738   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3739   \int_if_zero:nT { \l_@@_last_col_int }
3740   {
3741     \bool_set_true:N \l_@@_last_col_without_value_bool
3742     \int_set:Nn \l_@@_last_col_int { -1 }
3743   }
3744   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3745   \bool_lazy_or:nnT
3746   { \clist_if_empty_p:N \l_@@_vlines_clist }
3747   { \l_@@_except_borders_bool }
3748   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3749   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3750 }
3751 { \endNiceArray }

```

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

```

3752 \cs_new_protected:Npn \@@_NotEmpty:
3753 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3754 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3755 {

```

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.

```

3756   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3757   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3758   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3759   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3760   \tl_if_empty:NF \l_@@_short_caption_tl
3761   {
3762     \tl_if_empty:NT \l_@@_caption_tl
3763     {
3764       \@@_error_or_warning:n { short-caption-without~caption }
3765       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3766     }
3767   }
3768   \tl_if_empty:NF \l_@@_label_tl
3769   {

```

```

3770     \tl_if_empty:NT \l_@@_caption_tl
3771     { \@@_error_or_warning:n { label~without~caption } }
3772   }
3773   \NewDocumentEnvironment { TabularNote } { b }
3774   {
3775     \bool_if:NTF \l_@@_in_code_after_bool
3776     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3777     {
3778       \tl_if_empty:NF \g_@@_tabularnote_tl
3779       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3780       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3781     }
3782   }
3783   { }
3784   \@@_settings_for_tabular:
3785   \NiceArray { #2 }
3786 }
3787 { \endNiceArray }
3788 \cs_new_protected:Npn \@@_settings_for_tabular:
3789 {
3790   \bool_set_true:N \l_@@_tabular_bool
3791   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3792   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3793   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3794 }

3795 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3796 {
3797   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3798   \dim_set:Nn \l_@@_width_dim { #1 }
3799   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3800   \@@_settings_for_tabular:
3801   \NiceArray { #3 }
3802 }
3803 {
3804   \endNiceArray
3805   \int_if_zero:nT { \g_@@_total_X_weight_int }
3806   { \@@_error:n { NiceTabularX~without~X } }
3807 }

3808 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3809 {
3810   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3811   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3812   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3813   \@@_settings_for_tabular:
3814   \NiceArray { #3 }
3815 }
3816 { \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`).

```

3817 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3818 {
3819   \bool_lazy_all:nT
3820   {

```

```

3821     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3822     { \l_@@_hvlines_bool }
3823     { ! \g_@@_delims_bool }
3824     { ! \l_@@_except_borders_bool }
3825   }
3826   {
3827     \bool_set_true:N \l_@@_except_borders_bool
3828     \clist_if_empty:NF \l_@@_corners_clist
3829     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3830     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3831     {
3832       \@@_stroke_block:nnn
3833       {
3834         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3835         draw = \l_@@_rules_color_tl
3836       }
3837       { 1-1 }
3838       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3839     }
3840   }
3841 }

```

```

3842 \cs_new_protected:Npn \@@_after_array:
3843 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` 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`.

```

3844   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3845   \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.

```

3846   \bool_if:NT \g_@@_last_col_found_bool
3847   { \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`.

```

3848   \bool_if:NT \l_@@_last_col_without_value_bool
3849   { \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.

```

3850   \bool_if:NT \l_@@_last_row_without_value_bool
3851   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3852   \tl_gput_right:Ne \g_@@_aux_tl
3853   {
3854     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3855     {
3856       \int_use:N \l_@@_first_row_int ,
3857       \int_use:N \c@iRow ,
3858       \int_use:N \g_@@_row_total_int ,
3859       \int_use:N \l_@@_first_col_int ,
3860       \int_use:N \c@jCol ,
3861       \int_use:N \g_@@_col_total_int
3862     }
3863   }

```


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`).

```

3864 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3865 {
3866   \tl_gput_right:Ne \g_@@_aux_tl
3867   {
3868     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3869     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3870   }
3871 }
3872 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3873 {
3874   \tl_gput_right:Ne \g_@@_aux_tl
3875   {
3876     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3877     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3878     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3879     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3880   }
3881 }

```

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

```

3882 \@@_create_diag_nodes:

```

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

```

3883 \pgfpicture
3884 \@@_create_aliases_last:
3885 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3886 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹². 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.

```

3887 \bool_if:NT \l_@@_parallelize_diags_bool
3888 {
3889   \int_gzero:N \g_@@_ddots_int
3890   \int_gzero:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_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 Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3891   \dim_gzero:N \g_@@_delta_x_one_dim
3892   \dim_gzero:N \g_@@_delta_y_one_dim
3893   \dim_gzero:N \g_@@_delta_x_two_dim
3894   \dim_gzero:N \g_@@_delta_y_two_dim
3895 }
3896 \bool_set_false:N \l_@@_initial_open_bool
3897 \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.

```

3898 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3899 \@@_draw_dotted_lines:

```

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

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.

```

3900   \clist_if_empty:NF \l_@@_corners_cells_clist
3901   {
3902     \bool_if:NTF \l_@@_no_cell_nodes_bool
3903     { \@@_error:n { corners~with~no~cell~nodes } }
3904     { \@@_compute_corners: }
3905   }

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3906   \@@_adjust_pos_of_blocks_seq:
3907   \@@_deal_with_rounded_corners:
3908   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3909   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3910   \IfPackageLoadedT { tikz }
3911   {
3912     \tikzset
3913     {
3914       every-picture / .style =
3915       {
3916         overlay ,
3917         remember~picture ,
3918         name~prefix = \@@_env: -
3919       }
3920     }
3921   }
3922   \bool_if:NT \c_@@_recent_array_bool
3923   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3924   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3925   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3926   \cs_set_eq:NN \OverBrace \@@_OverBrace
3927   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3928   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3929   \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.

```

3930   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3931   \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.

```

3932   \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).

```

3933   \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`).

```

3934   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3935   { \@@_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:`.

```

3936 \bool_set_true:N \l_@@_in_code_after_bool
3937 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3938 \scan_stop:
3939 \tl_gclear:N \g_nicematrix_code_after_tl
3940 \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.

```

3941 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3942 \tl_if_empty:NF \g_@@_pre_code_before_tl
3943 {
3944   \tl_gput_right:Ne \g_@@_aux_tl
3945   {
3946     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3947     { \exp_not:o \g_@@_pre_code_before_tl }
3948   }
3949   \tl_gclear:N \g_@@_pre_code_before_tl
3950 }
3951 \tl_if_empty:NF \g_nicematrix_code_before_tl
3952 {
3953   \tl_gput_right:Ne \g_@@_aux_tl
3954   {
3955     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3956     { \exp_not:o \g_nicematrix_code_before_tl }
3957   }
3958   \tl_gclear:N \g_nicematrix_code_before_tl
3959 }

3960 \str_gclear:N \g_@@_name_env_str
3961 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. 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`.

```

3962 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3963 }

3964 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3965 {
3966   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3967   \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 corre-
spond to the options xdots/shorten-start and xdots/shorten-end available to the user.

3968   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3969   { 0.6 \l_@@_xdots_shorten_start_dim }
3970   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3971   { 0.6 \l_@@_xdots_shorten_end_dim }
3972 }

```

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`.

```

3973 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3974 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

3975 \cs_new_protected:Npn \@@_create_alias_nodes:
3976 {
3977   \int_step_inline:nn { \c@iRow }
3978   {
3979     \pgfnodealias
3980     { \l_@@_name_str - ##1 - last }
3981     { \@@_env: - ##1 - \int_use:N \c@jCol }
3982   }
3983   \int_step_inline:nn { \c@jCol }
3984   {
3985     \pgfnodealias
3986     { \l_@@_name_str - last - ##1 }
3987     { \@@_env: - \int_use:N \c@iRow - ##1 }
3988   }
3989   \pgfnodealias % added 2025-04-05
3990   { \l_@@_name_str - last - last }
3991   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3992 }

```

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).

```

3993 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3994 {
3995   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3996   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3997 }

```

The following command must *not* be protected.

```

3998 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3999 {
4000   { #1 }
4001   { #2 }
4002   {
4003     \int_compare:nNnTF { #3 } > { 98 }
4004     { \int_use:N \c@iRow }
4005     { #3 }
4006   }
4007   {
4008     \int_compare:nNnTF { #4 } > { 98 }
4009     { \int_use:N \c@jCol }
4010     { #4 }
4011   }
4012   { #5 }
4013 }

```

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).

```

4014 \hook_gput_code:nnn { begindocument } { . }
4015 {
4016   \cs_new_protected:Npe \@@_draw_dotted_lines:
4017   {
4018     \c_@@_pgfortikzpicture_tl
4019     \@@_draw_dotted_lines_i:
4020     \c_@@_endpgfortikzpicture_tl
4021   }
4022 }

```

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

```

4023 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4024 {
4025   \pgfrememberpicturepositiononpagetrue
4026   \pgf@relevantforpicturesizefalse
4027   \g_@@_HVdotsfor_lines_tl
4028   \g_@@_Vdots_lines_tl
4029   \g_@@_Ddots_lines_tl
4030   \g_@@_Iddots_lines_tl
4031   \g_@@_Cdots_lines_tl
4032   \g_@@_Ldots_lines_tl
4033 }

4034 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4035 {
4036   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4037   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4038 }

```

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

```

4039 \pgfdeclareshape { @@_diag_node }
4040 {
4041   \savedanchor { \five }
4042   {
4043     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4044     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4045   }
4046   \anchor { 5 } { \five }
4047   \anchor { center } { \pgfpointorigin }
4048   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4049   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4050   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4051   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4052   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4053   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4054   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4055   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4056   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4057   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4058 }

```

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).

```

4059 \cs_new_protected:Npn \@@_create_diag_nodes:
4060 {
4061   \pgfpicture
4062   \pgfrememberpicturepositiononpagetrue
4063   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4064   {
4065     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4066     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4067     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4068     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4069     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4070     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4071     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4072     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4073     \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.

```

4074     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4075     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4076     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4077     \str_if_empty:NF \l_@@_name_str
4078     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4079 }

```

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

```

4080     \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4081     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4082     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4083     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4084     \pgfcoordinate
4085     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4086     \pgfnodealias
4087     { \@@_env: - last }
4088     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4089     \str_if_empty:NF \l_@@_name_str
4090     {
4091         \pgfnodealias
4092         { \l_@@_name_str - \int_use:N \l_tmpa_int }
4093         { \@@_env: - \int_use:N \l_tmpa_int }
4094         \pgfnodealias
4095         { \l_@@_name_str - last }
4096         { \@@_env: - last }
4097     }
4098     \endpgfpicture
4099 }

```

16 We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

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

The command `\@@_find_extremities_of_line: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.

```
4100 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4101 {
```

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

```
4102 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4103 \int_set:Nn \l_@@_initial_i_int { #1 }
4104 \int_set:Nn \l_@@_initial_j_int { #2 }
4105 \int_set:Nn \l_@@_final_i_int { #1 }
4106 \int_set:Nn \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.

```
4107 \bool_set_false:N \l_@@_stop_loop_bool
4108 \bool_do_until:Nn \l_@@_stop_loop_bool
4109 {
4110 \int_add:Nn \l_@@_final_i_int { #3 }
4111 \int_add:Nn \l_@@_final_j_int { #4 }
4112 \bool_set_false:N \l_@@_final_open_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.

```
4113 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4114 \if_int_compare:w #3 = \c_one_int
4115 \bool_set_true:N \l_@@_final_open_bool
4116 \else:
4117 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4118 \bool_set_true:N \l_@@_final_open_bool
4119 \fi:
4120 \fi:
4121 \else:
4122 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4123 \if_int_compare:w #4 = -1
4124 \bool_set_true:N \l_@@_final_open_bool
4125 \fi:
4126 \else:
4127 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4128 \if_int_compare:w #4 = \c_one_int
4129 \bool_set_true:N \l_@@_final_open_bool
4130 \fi:
4131 \fi:
4132 \fi:
4133 \fi:
4134 \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.

```
4135 {
```

We do a step backwards.

```
4136 \int_sub:Nn \l_@@_final_i_int { #3 }
4137 \int_sub:Nn \l_@@_final_j_int { #4 }
4138 \bool_set_true:N \l_@@_stop_loop_bool
4139 }
```

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`.

```
4140 {
4141 \cs_if_exist:cTF
4142 {
4143 @@ _ dotted _
```

```

4144         \int_use:N \l_@@_final_i_int -
4145         \int_use:N \l_@@_final_j_int
4146     }
4147     {
4148         \int_sub:Nn \l_@@_final_i_int { #3 }
4149         \int_sub:Nn \l_@@_final_j_int { #4 }
4150         \bool_set_true:N \l_@@_final_open_bool
4151         \bool_set_true:N \l_@@_stop_loop_bool
4152     }
4153     {
4154         \cs_if_exist:cTF
4155         {
4156             pgf @ sh @ ns @ \@@_env:
4157             - \int_use:N \l_@@_final_i_int
4158             - \int_use:N \l_@@_final_j_int
4159         }
4160         { \bool_set_true:N \l_@@_stop_loop_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.

```

4161         {
4162             \cs_set_nopar:cpn
4163             {
4164                 @@ _ dotted _
4165                 \int_use:N \l_@@_final_i_int -
4166                 \int_use:N \l_@@_final_j_int
4167             }
4168             { }
4169         }
4170     }
4171 }
4172 }

```

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

```

4173     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4174     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4175     \bool_do_until:Nn \l_@@_stop_loop_bool
4176     {
4177         \int_sub:Nn \l_@@_initial_i_int { #3 }
4178         \int_sub:Nn \l_@@_initial_j_int { #4 }
4179         \bool_set_false:N \l_@@_initial_open_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.

```

4180         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4181         \if_int_compare:w #3 = \c_one_int
4182         \bool_set_true:N \l_@@_initial_open_bool
4183         \else:

```

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

```

4184         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4185         \bool_set_true:N \l_@@_initial_open_bool
4186         \fi:
4187         \fi:
4188         \else:
4189         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4190         \if_int_compare:w #4 = \c_one_int

```



```

4191         \bool_set_true:N \l_@@_initial_open_bool
4192     \fi:
4193 \else:
4194     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4195         \if_int_compare:w #4 = -1
4196             \bool_set_true:N \l_@@_initial_open_bool
4197         \fi:
4198     \fi:
4199 \fi:
4200 \fi:
4201 \bool_if:NTF \l_@@_initial_open_bool
4202 {
4203     \int_add:Nn \l_@@_initial_i_int { #3 }
4204     \int_add:Nn \l_@@_initial_j_int { #4 }
4205     \bool_set_true:N \l_@@_stop_loop_bool
4206 }
4207 {
4208     \cs_if_exist:cTF
4209     {
4210         @@ _ dotted _
4211         \int_use:N \l_@@_initial_i_int -
4212         \int_use:N \l_@@_initial_j_int
4213     }
4214     {
4215         \int_add:Nn \l_@@_initial_i_int { #3 }
4216         \int_add:Nn \l_@@_initial_j_int { #4 }
4217         \bool_set_true:N \l_@@_initial_open_bool
4218         \bool_set_true:N \l_@@_stop_loop_bool
4219     }
4220     {
4221         \cs_if_exist:cTF
4222         {
4223             pgf @ sh @ ns @ \@@_env:
4224             - \int_use:N \l_@@_initial_i_int
4225             - \int_use:N \l_@@_initial_j_int
4226         }
4227         { \bool_set_true:N \l_@@_stop_loop_bool }
4228         {
4229             \cs_set_nopar:cpn
4230             {
4231                 @@ _ dotted _
4232                 \int_use:N \l_@@_initial_i_int -
4233                 \int_use:N \l_@@_initial_j_int
4234             }
4235             { }
4236         }
4237     }
4238 }
4239 }

```

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.

```

4240 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4241 {
4242     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4243     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4244     { \int_use:N \l_@@_final_i_int }
4245     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4246     { } % for the name of the block
4247 }
4248 }

```

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 know 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.

```

4249 \cs_new_protected:Npn \@@_open_shorten:
4250 {
4251   \bool_if:NT \l_@@_initial_open_bool
4252     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4253   \bool_if:NT \l_@@_final_open_bool
4254     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4255 }

```

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).

```

4256 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4257 {
4258   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4259   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4260   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4261   \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`.

```

4262   \seq_if_empty:NF \g_@@_submatrix_seq
4263   {
4264     \seq_map_inline:Nn \g_@@_submatrix_seq
4265       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4266   }
4267 }

```

#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 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.

```

4268 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4269 {
4270   \if_int_compare:w #3 > #1
4271   \else:
4272     \if_int_compare:w #1 > #5

```

```

4273 \else:
4274 \if_int_compare:w #4 > #2
4275 \else:
4276 \if_int_compare:w #2 > #6
4277 \else:
4278 \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4279 \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4280 \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4281 \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4282 \fi:
4283 \fi:
4284 \fi:
4285 \fi:
4286 }

4287 \cs_new_protected:Npn \@@_set_initial_coords:
4288 {
4289 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4290 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4291 }
4292 \cs_new_protected:Npn \@@_set_final_coords:
4293 {
4294 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4295 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4296 }
4297 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4298 {
4299 \pgfpointanchor
4300 {
4301 \@@_env:
4302 - \int_use:N \l_@@_initial_i_int
4303 - \int_use:N \l_@@_initial_j_int
4304 }
4305 { #1 }
4306 \@@_set_initial_coords:
4307 }
4308 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4309 {
4310 \pgfpointanchor
4311 {
4312 \@@_env:
4313 - \int_use:N \l_@@_final_i_int
4314 - \int_use:N \l_@@_final_j_int
4315 }
4316 { #1 }
4317 \@@_set_final_coords:
4318 }
4319 \cs_new_protected:Npn \@@_open_x_initial_dim:
4320 {
4321 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4322 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4323 {
4324 \cs_if_exist:cT
4325 { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4326 {
4327 \pgfpointanchor
4328 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4329 { west }
4330 \dim_set:Nn \l_@@_x_initial_dim
4331 { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4332 }
4333 }

```

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

```

4334 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4335 {
4336   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4337   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4338   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4339 }
4340 }
4341 \cs_new_protected:Npn \@@_open_x_final_dim:
4342 {
4343   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4344   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4345   {
4346     \cs_if_exist:cT
4347     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4348     {
4349       \pgfpointanchor
4350       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4351       { east }
4352       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4353       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4354     }
4355   }

```

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

```

4356 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4357 {
4358   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4359   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4360   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4361 }
4362 }

```

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.

```

4363 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4364 {
4365   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4366   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4367   {
4368     \@@_find_extremities_of_line: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.

```

4369 \group_begin:
4370 \@@_open_shorten:
4371 \int_if_zero:nTF { #1 }
4372 { \color { nicematrix-first-row } }
4373 {

```

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.

```

4374 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4375 { \color { nicematrix-last-row } }
4376 }
4377 \keys_set:nn { nicematrix / xdots } { #3 }
4378 \@@_color:o \l_@@_xdots_color_tl
4379 \@@_actually_draw_Ldots:
4380 \group_end:
4381 }
4382 }

```

The command `\@@_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.

```

4383 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4384 {
4385   \bool_if:NTF \l_@@_initial_open_bool
4386   {
4387     \@@_open_x_initial_dim:
4388     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4389     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4390   }
4391   { \@@_set_initial_coords_from_anchor:n { base-east } }
4392   \bool_if:NTF \l_@@_final_open_bool
4393   {
4394     \@@_open_x_final_dim:
4395     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4396     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4397   }
4398   { \@@_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.

```

4399   \bool_lazy_all:nTF
4400   {
4401     \l_@@_initial_open_bool
4402     \l_@@_final_open_bool
4403     { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4404   }
4405   {
4406     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4407     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4408   }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4409   {
4410     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4411     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4412   }
4413   \@@_draw_line:
4414 }

```

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.

```

4415 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4416 {
4417   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4418   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4419   {
4420     \@@_find_extremities_of_line: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.

```

4421 \group_begin:
4422 \@@_open_shorten:
4423 \int_if_zero:nTF { #1 }
4424 { \color { nicematrix-first-row } }
4425 {

```

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.

```

4426 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4427 { \color { nicematrix-last-row } }
4428 }
4429 \keys_set:nn { nicematrix / xdots } { #3 }
4430 \@@_color:o \l_@@_xdots_color_tl
4431 \@@_actually_draw_Cdots:
4432 \group_end:
4433 }
4434 }

```

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`.

```

4435 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4436 {
4437 \bool_if:NTF \l_@@_initial_open_bool
4438 { \@@_open_x_initial_dim: }
4439 { \@@_set_initial_coords_from_anchor:n { mid-east } }
4440 \bool_if:NTF \l_@@_final_open_bool
4441 { \@@_open_x_final_dim: }
4442 { \@@_set_final_coords_from_anchor:n { mid-west } }
4443 \bool_lazy_and:nnTF
4444 { \l_@@_initial_open_bool }
4445 { \l_@@_final_open_bool }
4446 {
4447 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4448 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4449 \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4450 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4451 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4452 }
4453 {
4454 \bool_if:NT \l_@@_initial_open_bool
4455 { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4456 \bool_if:NT \l_@@_final_open_bool
4457 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4458 }
4459 \@@_draw_line:
4460 }
4461 \cs_new_protected:Npn \@@_open_y_initial_dim:
4462 {
4463 \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4464 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4465 {

```

```

4466 \cs_if_exist:cT
4467 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4468 {
4469     \pgfpointanchor
4470     { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4471     { north }
4472     \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4473     { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4474 }
4475 }
4476 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4477 {
4478     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4479     \dim_set:Nn \l_@@_y_initial_dim
4480     {
4481         \fp_to_dim:n
4482         {
4483             \pgf@y
4484             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4485         }
4486     }
4487 }
4488 }
4489 \cs_new_protected:Npn \@@_open_y_final_dim:
4490 {
4491     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4492     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4493     {
4494         \cs_if_exist:cT
4495         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4496         {
4497             \pgfpointanchor
4498             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4499             { south }
4500             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4501             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4502         }
4503     }
4504     \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4505     {
4506         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4507         \dim_set:Nn \l_@@_y_final_dim
4508         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4509     }
4510 }

```

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.

```

4511 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4512 {
4513     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4514     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4515     {
4516         \@@_find_extremities_of_line: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.

```

4517 \group_begin:
4518     \@@_open_shorten:
4519     \int_if_zero:nTF { #2 }
4520     { \color { nicematrix-first-col } }
4521     {
4522         \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4523         { \color { nicematrix-last-col } }

```

```

4524     }
4525     \keys_set:nn { nicematrix / xdots } { #3 }
4526     \@@_color:o \l_@@_xdots_color_tl
4527     \@@_actually_draw_Vdots:
4528   \group_end:
4529 }
4530 }

```

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.`

The following function is also used by `\Vdotsfor`.

```

4531 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4532 {

```

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

```

4533   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }

```

We have to determine the x -value of the vertical rule that we will have to draw.

```

4534   {
4535     \@@_open_y_initial_dim:
4536     \@@_open_y_final_dim:
4537     \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4538     {
4539       \@@_qpoint:n { col - 1 }
4540       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4541       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4542       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4543       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4544     }
4545     {
4546       \bool_lazy_and:nnTF
4547       { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4548       {
4549         \int_compare_p:nNn
4550         { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4551       }

```

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

```

4552     {
4553       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4554       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4555       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4556       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4557       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4558     }

```

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

```

4559     {
4560       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4561       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4562       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4563       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }

```



```

4564     }
4565   }
4566 }

```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

4567 {
4568   \bool_set_false:N \l_tmpa_bool
4569   \bool_if:NF \l_@@_initial_open_bool
4570   {
4571     \bool_if:NF \l_@@_final_open_bool
4572     {
4573       \@@_set_initial_coords_from_anchor:n { south-west }
4574       \@@_set_final_coords_from_anchor:n { north-west }
4575       \bool_set:Nn \l_tmpa_bool
4576       {
4577         \dim_compare_p:nNn
4578         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4579       }
4580     }
4581   }

```

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

```

4582   \bool_if:NTF \l_@@_initial_open_bool
4583   {
4584     \@@_open_y_initial_dim:
4585     \@@_set_final_coords_from_anchor:n { north }
4586     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4587   }
4588   {
4589     \@@_set_initial_coords_from_anchor:n { south }
4590     \bool_if:NTF \l_@@_final_open_bool
4591     { \@@_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.

```

4592   {
4593     \@@_set_final_coords_from_anchor:n { north }
4594     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4595     {
4596       \dim_set:Nn \l_@@_x_initial_dim
4597       {
4598         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4599         \l_@@_x_initial_dim \l_@@_x_final_dim
4600       }
4601     }
4602   }
4603 }
4604 }
4605 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4606 \@@_draw_line:
4607 }

```

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.

```

4608 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4609 {
4610   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4611   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4612   {
4613     \@@_find_extremities_of_line: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.

```

4614     \group_begin:
4615     \@@_open_shorten:
4616     \keys_set:nn { nicematrix / xdots } { #3 }
4617     \@@_color:o \l_@@_xdots_color_tl
4618     \@@_actually_draw_Ddots:
4619     \group_end:
4620 }
4621 }

```

The command `\@@_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.`

```

4622 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4623 {
4624   \bool_if:NTF \l_@@_initial_open_bool
4625   {
4626     \@@_open_y_initial_dim:
4627     \@@_open_x_initial_dim:
4628   }
4629   { \@@_set_initial_coords_from_anchor:n { south-east } }
4630   \bool_if:NTF \l_@@_final_open_bool
4631   {
4632     \@@_open_x_final_dim:
4633     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4634   }
4635   { \@@_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.

```

4636   \bool_if:NT \l_@@_parallelize_diags_bool
4637   {
4638     \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).

```

4639     \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 Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4640     {
4641       \dim_gset:Nn \g_@@_delta_x_one_dim
4642       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4643       \dim_gset:Nn \g_@@_delta_y_one_dim
4644       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4645     }

```

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`.

```

4646     {
4647       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4648       {
4649         \dim_set:Nn \l_@@_y_final_dim

```

```

4650         {
4651             \l_@@_y_initial_dim +
4652             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4653             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4654         }
4655     }
4656 }
4657 }
4658 \@@_draw_line:
4659 }

```

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.

```

4660 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4661 {
4662     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4663     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4664     {
4665         \@@_find_extremities_of_line: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.

```

4666         \group_begin:
4667         \@@_open_shorten:
4668         \keys_set:nn { nicematrix / xdots } { #3 }
4669         \@@_color:o \l_@@_xdots_color_tl
4670         \@@_actually_draw_Iddots:
4671     \group_end:
4672 }
4673 }

```

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.

```

4674 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4675 {
4676     \bool_if:NTF \l_@@_initial_open_bool
4677     {
4678         \@@_open_y_initial_dim:
4679         \@@_open_x_initial_dim:
4680     }
4681     { \@@_set_initial_coords_from_anchor:n { south-west } }
4682     \bool_if:NTF \l_@@_final_open_bool
4683     {
4684         \@@_open_y_final_dim:
4685         \@@_open_x_final_dim:
4686     }
4687     { \@@_set_final_coords_from_anchor:n { north-east } }
4688     \bool_if:NT \l_@@_parallelize_diags_bool
4689     {
4690         \int_gincr:N \g_@@_iddots_int
4691         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }

```

```

4692     {
4693         \dim_gset:Nn \g_@@_delta_x_two_dim
4694         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4695         \dim_gset:Nn \g_@@_delta_y_two_dim
4696         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4697     }
4698     {
4699         \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4700         {
4701             \dim_set:Nn \l_@@_y_final_dim
4702             {
4703                 \l_@@_y_initial_dim +
4704                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4705                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4706             }
4707         }
4708     }
4709 }
4710 \@@_draw_line:
4711 }

```

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`

```

4712 \cs_new_protected:Npn \@@_draw_line:
4713 {
4714     \pgfrememberpicturepositiononpagetrue
4715     \pgf@relevantforpicturesizefalse
4716     \bool_lazy_or:nnTF
4717     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4718     { \l_@@_dotted_bool }
4719     { \@@_draw_standard_dotted_line: }
4720     { \@@_draw_unstandard_dotted_line: }
4721 }

```

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.

```

4722 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4723 {
4724     \begin { scope }
4725     \@@_draw_unstandard_dotted_line:o
4726     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4727 }

```

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.

```

4728 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4729 {
4730   \@@_draw_unstandard_dotted_line:nooo
4731   { #1 }
4732   \l_@@_xdots_up_tl
4733   \l_@@_xdots_down_tl
4734   \l_@@_xdots_middle_tl
4735 }
4736 \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.

```

4737 \hook_gput_code:nnn { begindocument } { . }
4738 {
4739   \IfPackageLoadedT { tikz }
4740   {
4741     \tikzset
4742     {
4743       @@_node_above / .style = { sloped , above } ,
4744       @@_node_below / .style = { sloped , below } ,
4745       @@_node_middle / .style =
4746       {
4747         sloped ,
4748         inner~sep = \c_@@_innersep_middle_dim
4749       }
4750     }
4751   }
4752 }

4753 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4754 {

```

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 ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4755 \dim_zero_new:N \l_@@_l_dim
4756 \dim_set:Nn \l_@@_l_dim
4757 {
4758   \fp_to_dim:n
4759   {
4760     sqrt
4761     (
4762       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4763       +
4764       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4765     )
4766   }
4767 }

```

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.

```

4768 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4769 {
4770   \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }

```

```

4771 \@@_draw_unstandard_dotted_line_i:
4772 }

```

If the key `xdots/horizontal-labels` has been used.

```

4773 \bool_if:NT \l_@@_xdots_h_labels_bool
4774 {
4775   \tikzset
4776   {
4777     @@_node_above / .style = { auto = left } ,
4778     @@_node_below / .style = { auto = right } ,
4779     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4780   }
4781 }
4782 \tl_if_empty:nF { #4 }
4783 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4784 \draw
4785 [ #1 ]
4786 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4787 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4788 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4789 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4790 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4791 \end { scope }
4792 }
4793 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4794 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4795 {
4796   \dim_set:Nn \l_tmpa_dim
4797   {
4798     \l_@@_x_initial_dim
4799     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4800     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4801   }
4802   \dim_set:Nn \l_tmpb_dim
4803   {
4804     \l_@@_y_initial_dim
4805     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4806     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4807   }
4808   \dim_set:Nn \l_@@_tmpc_dim
4809   {
4810     \l_@@_x_final_dim
4811     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4812     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4813   }
4814   \dim_set:Nn \l_@@_tmpd_dim
4815   {
4816     \l_@@_y_final_dim
4817     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4818     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4819   }
4820   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4821   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4822   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4823   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4824 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4825 \cs_new_protected:Npn \@@_draw_standard_dotted_line:

```

```

4826 {
4827   \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.
4828   \dim_zero_new:N \l_@@_l_dim
4829   \dim_set:Nn \l_@@_l_dim
4830   {
4831     \fp_to_dim:n
4832     {
4833       sqrt
4834       (
4835         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4836         +
4837         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4838       )
4839     }
4840   }

```

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.

```

4841   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4842   {
4843     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4844     { \@@_draw_standard_dotted_line_i: }
4845   }
4846   \group_end:
4847   \bool_lazy_all:nF
4848   {
4849     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4850     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4851     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4852   }
4853   { \@@_labels_standard_dotted_line: }
4854 }
4855 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4856 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4857 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4858   \int_set:Nn \l_tmpa_int
4859   {
4860     \dim_ratio:nn
4861     {
4862       \l_@@_l_dim
4863       - \l_@@_xdots_shorten_start_dim
4864       - \l_@@_xdots_shorten_end_dim
4865     }
4866     { \l_@@_xdots_inter_dim }
4867   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4868   \dim_set:Nn \l_tmpa_dim
4869   {
4870     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4871     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4872   }
4873   \dim_set:Nn \l_tmpb_dim
4874   {
4875     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

4876     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4877 }

```

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.

```

4878 \dim_gadd:Nn \l_@@_x_initial_dim
4879 {
4880   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4881   \dim_ratio:nn
4882   {
4883     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4884     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4885   }
4886   { 2 \l_@@_l_dim }
4887 }
4888 \dim_gadd:Nn \l_@@_y_initial_dim
4889 {
4890   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4891   \dim_ratio:nn
4892   {
4893     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4894     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4895   }
4896   { 2 \l_@@_l_dim }
4897 }
4898 \pgf@relevantforpicturesizefalse
4899 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4900 {
4901   \pgfpathcircle
4902   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4903   { \l_@@_xdots_radius_dim }
4904   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4905   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4906 }
4907 \pgfusepathqfill
4908 }

```

```

4909 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4910 {
4911   \pgfscope
4912   \pgftransformshift
4913   {
4914     \pgfpointlineattime { 0.5 }
4915     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4916     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4917   }
4918   \fp_set:Nn \l_tmpa_fp
4919   {
4920     atand
4921     (
4922       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4923       \l_@@_x_final_dim - \l_@@_x_initial_dim
4924     )
4925   }
4926   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4927   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4928   \tl_if_empty:NF \l_@@_xdots_middle_tl
4929   {
4930     \begin { pgfscope }
4931     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4932     \pgfnode
4933     { rectangle }
4934     { center }

```



```

4935     {
4936         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4937         {
4938             \c_math_toggle_token
4939             \scriptstyle \l_@@_xdots_middle_tl
4940             \c_math_toggle_token
4941         }
4942     }
4943     { }
4944     {
4945         \pgfsetfillcolor { white }
4946         \pgfusepath { fill }
4947     }
4948     \end { pgfscope }
4949 }
4950 \tl_if_empty:NF \l_@@_xdots_up_tl
4951 {
4952     \pgfnode
4953     { rectangle }
4954     { south }
4955     {
4956         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4957         {
4958             \c_math_toggle_token
4959             \scriptstyle \l_@@_xdots_up_tl
4960             \c_math_toggle_token
4961         }
4962     }
4963     { }
4964     { \pgfusepath { } }
4965 }
4966 \tl_if_empty:NF \l_@@_xdots_down_tl
4967 {
4968     \pgfnode
4969     { rectangle }
4970     { north }
4971     {
4972         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4973         {
4974             \c_math_toggle_token
4975             \scriptstyle \l_@@_xdots_down_tl
4976             \c_math_toggle_token
4977         }
4978     }
4979     { }
4980     { \pgfusepath { } }
4981 }
4982 \endpgfscope
4983 }

```

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.

```
4984 \hook_gput_code:nnn { begindocument } { . }
4985 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
4986 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4987 \cs_new_protected:Npn \@@_Ldots:
4988 { \@@_collect_options:n { \@@_Ldots_i } }
4989 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4990 {
4991   \int_if_zero:nTF { \c@jCol }
4992   { \@@_error:nn { in~first~col } { \Ldots } }
4993   {
4994     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
4995     { \@@_error:nn { in~last~col } { \Ldots } }
4996     {
4997       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
4998       { #1 , down = #2 , up = #3 , middle = #4 }
4999     }
5000   }
5001   \bool_if:NF \l_@@_nullify_dots_bool
5002   { \phantom { \ensuremath { \@@_old_ldots: } } }
5003   \bool_gset_true:N \g_@@_empty_cell_bool
5004 }
```

```
5005 \cs_new_protected:Npn \@@_Cdots:
5006 { \@@_collect_options:n { \@@_Cdots_i } }
5007 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5008 {
5009   \int_if_zero:nTF { \c@jCol }
5010   { \@@_error:nn { in~first~col } { \Cdots } }
5011   {
5012     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5013     { \@@_error:nn { in~last~col } { \Cdots } }
5014     {
5015       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5016       { #1 , down = #2 , up = #3 , middle = #4 }
5017     }
5018   }
5019   \bool_if:NF \l_@@_nullify_dots_bool
5020   { \phantom { \ensuremath { \@@_old_cdots: } } }
5021   \bool_gset_true:N \g_@@_empty_cell_bool
5022 }
```

```
5023 \cs_new_protected:Npn \@@_Vdots:
5024 { \@@_collect_options:n { \@@_Vdots_i } }
5025 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5026 {
5027   \int_if_zero:nTF { \c@iRow }
5028   { \@@_error:nn { in~first~row } { \Vdots } }
5029   {
5030     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5031     { \@@_error:nn { in~last~row } { \Vdots } }
5032     {
5033       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5034       { #1 , down = #2 , up = #3 , middle = #4 }
5035     }
5036   }
5037   \bool_if:NF \l_@@_nullify_dots_bool
5038   { \phantom { \ensuremath { \@@_old_vdots: } } }
```

```

5039     \bool_gset_true:N \g_@@_empty_cell_bool
5040 }

5041 \cs_new_protected:Npn \@@_Ddots:
5042 { \@@_collect_options:n { \@@_Ddots_i } }
5043 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5044 {
5045     \int_case:nnF \c@iRow
5046     {
5047         0 { \@@_error:nn { in~first~row } { \Ddots } }
5048         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5049     }
5050     {
5051         \int_case:nnF \c@jCol
5052         {
5053             0 { \@@_error:nn { in~first~col } { \Ddots } }
5054             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5055         }
5056         {
5057             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5058             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5059             { #1 , down = #2 , up = #3 , middle = #4 }
5060         }
5061     }
5062 }
5063 \bool_if:NF \l_@@_nullify_dots_bool
5064 { \phantom { \ensuremath { \@@_old_ddots: } } }
5065 \bool_gset_true:N \g_@@_empty_cell_bool
5066 }

5067 \cs_new_protected:Npn \@@_Iddots:
5068 { \@@_collect_options:n { \@@_Iddots_i } }
5069 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5070 {
5071     \int_case:nnF \c@iRow
5072     {
5073         0 { \@@_error:nn { in~first~row } { \Iddots } }
5074         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5075     }
5076     {
5077         \int_case:nnF \c@jCol
5078         {
5079             0 { \@@_error:nn { in~first~col } { \Iddots } }
5080             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5081         }
5082         {
5083             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5084             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5085             { #1 , down = #2 , up = #3 , middle = #4 }
5086         }
5087     }
5088     \bool_if:NF \l_@@_nullify_dots_bool
5089     { \phantom { \ensuremath { \@@_old_iddots: } } }
5090     \bool_gset_true:N \g_@@_empty_cell_bool
5091 }
5092 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5093 \keys_define:nn { nicematrix / Ddots }
5094 {

```

```

5095     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5096     draw-first .default:n = true ,
5097     draw-first .value_forbidden:n = true
5098 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5099 \cs_new_protected:Npn \@@_Hspace:
5100 {
5101   \bool_gset_true:N \g_@@_empty_cell_bool
5102   \hspace
5103 }

```

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`.

```

5104 \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`.

```

5105 \cs_new:Npn \@@_Hdotsfor:
5106 {
5107   \bool_lazy_and:nnTF
5108     { \int_if_zero_p:n { \c@jCol } }
5109     { \int_if_zero_p:n { \l_@@_first_col_int } }
5110     {
5111       \bool_if:NTF \g_@@_after_col_zero_bool
5112       {
5113         \multicolumn { 1 } { c } { }
5114         \@@_Hdotsfor_i:
5115       }
5116       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5117     }
5118   {
5119     \multicolumn { 1 } { c } { }
5120     \@@_Hdotsfor_i:
5121   }
5122 }

```

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:`).

```

5123 \hook_gput_code:nnn { begindocument } { . }
5124 {

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5125   \cs_new_protected:Npn \@@_Hdotsfor_i:
5126     { \@@_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`).

```

5127   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { } { } { } }
5128   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5129     {
5130     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5131     {
5132       \@@_Hdotsfor:nnnn
5133       { \int_use:N \c@iRow }
5134       { \int_use:N \c@jCol }
5135       { #2 }
5136       {
5137         #1 , #3 ,

```

```

5138         down = \exp_not:n { #4 } ,
5139         up = \exp_not:n { #5 } ,
5140         middle = \exp_not:n { #6 }
5141     }
5142 }
5143 \prg_replicate:nn { #2 - 1 }
5144 {
5145     &
5146     \multicolumn { 1 } { c } { }
5147     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5148 }
5149 }
5150 }

```

```

5151 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5152 {
5153     \bool_set_false:N \l_@@_initial_open_bool
5154     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5155     \int_set:Nn \l_@@_initial_i_int { #1 }
5156     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5157     \int_compare:nNnTF { #2 } = { \c_one_int }
5158     {
5159         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5160         \bool_set_true:N \l_@@_initial_open_bool
5161     }
5162     {
5163         \cs_if_exist:cTF
5164         {
5165             pgf @ sh @ ns @ \@@_env:
5166             - \int_use:N \l_@@_initial_i_int
5167             - \int_eval:n { #2 - 1 }
5168         }
5169         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5170         {
5171             \int_set:Nn \l_@@_initial_j_int { #2 }
5172             \bool_set_true:N \l_@@_initial_open_bool
5173         }
5174     }
5175     \int_compare:nNnTF { #2 + #3 - 1 } = { \c_jCol }
5176     {
5177         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5178         \bool_set_true:N \l_@@_final_open_bool
5179     }
5180     {
5181         \cs_if_exist:cTF
5182         {
5183             pgf @ sh @ ns @ \@@_env:
5184             - \int_use:N \l_@@_final_i_int
5185             - \int_eval:n { #2 + #3 }
5186         }
5187         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5188         {
5189             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5190             \bool_set_true:N \l_@@_final_open_bool
5191         }
5192     }
5193     \group_begin:
5194     \@@_open_shorten:
5195     \int_if_zero:nTF { #1 }
5196     { \color { nicematrix-first-row } }

```

```

5197     {
5198       \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5199       { \color { nicematrix-last-row } }
5200     }
5201     \keys_set:nn { nicematrix / xdots } { #4 }
5202     \@@_color:o \l_@@_xdots_color_tl
5203     \@@_actually_draw_Ldots:
5204     \group_end:

```

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_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5205     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5206     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5207   }

```

```

5208 \hook_gput_code:nnn { begindocument } { . }
5209 {
5210   \cs_new_protected:Npn \@@_Vdotsfor:
5211     { \@@_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`).

```

5212   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } { } } }
5213   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5214   {
5215     \bool_gset_true:N \g_@@_empty_cell_bool
5216     \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5217     {
5218       \@@_Vdotsfor:nnnn
5219       { \int_use:N \c@iRow }
5220       { \int_use:N \c@jCol }
5221       { #2 }
5222       {
5223         #1 , #3 ,
5224         down = \exp_not:n { #4 } ,
5225         up = \exp_not:n { #5 } ,
5226         middle = \exp_not:n { #6 }
5227       }
5228     }
5229   }
5230 }

```

```

5231 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5232 {
5233   \bool_set_false:N \l_@@_initial_open_bool
5234   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5235   \int_set:Nn \l_@@_initial_j_int { #2 }
5236   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

5237   \int_compare:nNnTF { #1 } = { \c_one_int }
5238   {
5239     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5240     \bool_set_true:N \l_@@_initial_open_bool
5241   }
5242   {
5243     \cs_if_exist:cTF
5244     {
5245       pgf @ sh @ ns @ \@@_env:

```

```

5246         - \int_eval:n { #1 - 1 }
5247         - \int_use:N \l_@@_initial_j_int
5248     }
5249     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5250     {
5251         \int_set:Nn \l_@@_initial_i_int { #1 }
5252         \bool_set_true:N \l_@@_initial_open_bool
5253     }
5254 }
5255 \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5256 {
5257     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5258     \bool_set_true:N \l_@@_final_open_bool
5259 }
5260 {
5261     \cs_if_exist:cTF
5262     {
5263         pgf @ sh @ ns @ \@@_env:
5264         - \int_eval:n { #1 + #3 }
5265         - \int_use:N \l_@@_final_j_int
5266     }
5267     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5268     {
5269         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5270         \bool_set_true:N \l_@@_final_open_bool
5271     }
5272 }
5273 \group_begin:
5274 \@@_open_shorten:
5275 \int_if_zero:nTF { #2 }
5276 { \color { nicematrix-first-col } }
5277 {
5278     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5279     { \color { nicematrix-last-col } }
5280 }
5281 \keys_set:nn { nicematrix / xdots } { #4 }
5282 \@@_color:o \l_@@_xdots_color_tl
5283 \@@_actually_draw_Vdots:
5284 \group_end:

```

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_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5285     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5286     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5287 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5288 \NewDocumentCommand \@@_rotate: { 0 { } }
5289 {
5290     \bool_gset_true:N \g_@@_rotate_bool
5291     \keys_set:nn { nicematrix / rotate } { #1 }
5292     \ignorespaces
5293 }
5294 \keys_define:nn { nicematrix / rotate }
5295 {
5296     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5297     c .value_forbidden:n = true ,
5298     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5299 }

```

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).¹⁴

```

5300 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5301 {
5302   \tl_if_empty:nTF { #2 }
5303     { #1 }
5304     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5305 }
5306 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5307 { \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).

```

5308 \hook_gput_code:nnn { begindocument } { . }
5309 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5310   \tl_set_rescan:Nnn \l_tmpa_tl { }
5311   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5312   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5313   {
5314     \group_begin:
5315     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5316     \@@_color:o \l_@@_xdots_color_tl
5317     \use:e
5318     {
5319       \@@_line_i:nn
5320       { \@@_double_int_eval:n #2 - \q_stop }
5321       { \@@_double_int_eval:n #3 - \q_stop }
5322     }
5323     \group_end:
5324   }
5325 }
5326 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5327 {
5328   \bool_set_false:N \l_@@_initial_open_bool
5329   \bool_set_false:N \l_@@_final_open_bool
5330   \bool_lazy_or:nnTF
5331     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5332     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5333     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5334   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5335 }
```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.


```

5336 \hook_gput_code:nnn { begindocument } { . }
5337 {
5338   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5339   {

```

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:`.

```

5340     \c_@@_pgfortikzpicture_tl
5341     \@@_draw_line_iii:nn { #1 } { #2 }
5342     \c_@@_endpgfortikzpicture_tl
5343   }
5344 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5345 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5346 {
5347   \pgfrememberpicturepositiononpagetrue
5348   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5349   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5350   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5351   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5352   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5353   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5354   \@@_draw_line:
5355 }

```

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 curriification.

```

5356 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNtT { \c@iRow } < }
5357 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNfT { \c@jCol } < }

```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```

5358 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5359 {
5360   \tl_gput_right:Ne \g_@@_row_style_tl
5361   {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can’t be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5362     \exp_not:N
5363     \@@_if_row_less_than:nn
5364     { \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`).

```

5365     {
5366         \exp_not:N
5367         \@@_if_col_greater_than:nn
5368         { \int_eval:n { \c@jCol } }
5369         { \exp_not:n { #1 } \scan_stop: }
5370     }
5371 }
5372 }
5373 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5374 \keys_define:nn { nicematrix / RowStyle }
5375 {
5376     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5377     cell-space-top-limit .value_required:n = true ,
5378     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5379     cell-space-bottom-limit .value_required:n = true ,
5380     cell-space-limits .meta:n =
5381     {
5382         cell-space-top-limit = #1 ,
5383         cell-space-bottom-limit = #1 ,
5384     } ,
5385     color .tl_set:N = \l_@@_color_tl ,
5386     color .value_required:n = true ,
5387     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5388     bold .default:n = true ,
5389     nb-rows .code:n =
5390     \str_if_eq:eeTF { #1 } { * }
5391     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5392     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5393     nb-rows .value_required:n = true ,
5394     fill .tl_set:N = \l_@@_fill_tl ,
5395     fill .value_required:n = true ,
5396     opacity .tl_set:N = \l_@@_opacity_tl ,
5397     opacity .value_required:n = true ,
5398     rowcolor .tl_set:N = \l_@@_fill_tl ,
5399     rowcolor .value_required:n = true ,
5400     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5401     rounded-corners .default:n = 4 pt ,
5402     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5403 }

5404 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5405 {
5406     \group_begin:
5407     \tl_clear:N \l_@@_fill_tl
5408     \tl_clear:N \l_@@_opacity_tl
5409     \tl_clear:N \l_@@_color_tl
5410     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5411     \dim_zero:N \l_@@_rounded_corners_dim
5412     \dim_zero:N \l_tmpa_dim
5413     \dim_zero:N \l_tmpb_dim
5414     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` (of its alias `fill`) has been used.

```

5415     \tl_if_empty:NF \l_@@_fill_tl
5416     {
5417         \@@_add_opacity_to_fill:
5418         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5419         {

```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```
5420         \int_compare:nNnTF { \c@jCol } > { \c_one_int }
5421         {
```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row). The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5422         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5423         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5424         { \int_use:N \c@iRow - * }
5425         { \dim_use:N \l_@@_rounded_corners_dim }
```

Then, the other rows (if there are several rows).

```
5426         \int_compare:nNnT { \l_@@_key_nb_rows_int } > { \c_one_int }
5427         { \@@_rounded_from_row:n { \c@iRow + 1 } }
5428     }
```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```
5429         { \@@_rounded_from_row:n { \c@iRow } }
5430     }
5431 }
5432 \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```
5433     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5434     {
5435         \@@_put_in_row_style:e
5436         {
5437             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5438             {
```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```
5439                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5440                 { \dim_use:N \l_tmpa_dim }
5441             }
5442         }
5443     }
```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```
5444     \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5445     {
5446         \@@_put_in_row_style:e
5447         {
5448             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5449             {
5450                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5451                 { \dim_use:N \l_tmpb_dim }
5452             }
5453         }
5454     }
```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```
5455     \tl_if_empty:NF \l_@@_color_tl
5456     {
5457         \@@_put_in_row_style:e
5458         {
5459             \mode_leave_vertical:
5460             \@@_color:n { \l_@@_color_tl }
5461         }
5462     }
```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```
5463     \bool_if:NT \l_@@_bold_row_style_bool
5464     {
5465         \@@_put_in_row_style:n
```

```

5466     {
5467         \exp_not:n
5468         {
5469             \if_mode_math:
5470                 \c_math_toggle_token
5471                 \bfseries \boldmath
5472                 \c_math_toggle_token
5473             \else:
5474                 \bfseries \boldmath
5475             \fi:
5476         }
5477     }
5478 }
5479 \group_end:
5480 \g_@@_row_style_tl
5481 \ignorespaces
5482 }

```

The following commande must *not* be protected.

```

5483 \cs_new:Npn \@@_rounded_from_row:n #1
5484 {
5485     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5486     { \int_eval:n { #1 } - 1 }
5487     {
5488         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5489         - \exp_not:n { \int_use:N \c@jCol }
5490     }
5491     { \dim_use:N \l_@@_rounded_corners_dim }
5492 }

```

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).

```

5493 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5494 {

```

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.

```
5495 \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`.

```
5496 \str_if_in:nnF { #1 } { !! }
5497 {
5498 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5499 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5500 }
5501 \int_if_zero:nTF { \l_tmpa_int }
```

First, the case where the color is a *new* color (not in the sequence).

```
5502 {
5503 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5504 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5505 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5506 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5507 }
5508 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5509 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5510 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5511 {
5512 \dim_compare:nNt { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5513 }
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5514 \group_begin:
5515 \pgfsetcornersarced
5516 {
5517 \pgfpoint
5518 { \l_@@_tab_rounded_corners_dim }
5519 { \l_@@_tab_rounded_corners_dim }
5520 }
```

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.

```
5521 \bool_if:NTF \l_@@_hvlines_bool
5522 {
5523 \pgfpathrectanglecorners
5524 {
5525 \pgfpointadd
5526 { \@@_qpoint:n { row-1 } }
5527 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5528 }
5529 {
5530 \pgfpointadd
5531 {
5532 \@@_qpoint:n
5533 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5534 }
5535 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5536 }
```

```

5537     }
5538     {
5539         \pgfpathrectanglecorners
5540         { \@@_qpoint:n { row-1 } }
5541         {
5542             \pgfpointadd
5543             {
5544                 \@@_qpoint:n
5545                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5546             }
5547             { \pgfpoint \c_zero_dim \arrayrulewidth }
5548         }
5549     }
5550     \pgfusepath { clip }
5551     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5552     }
5553 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5554 \cs_new_protected:Npn \@@_actually_color:
5555 {
5556     \pgfpicture
5557     \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.

```

5558     \@@_clip_with_rounded_corners:
5559     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5560     {
5561         \int_compare:nNnTF { ##1 } = { \c_one_int }
5562         {
5563             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5564             \use:c { g_@@_color _ 1 _tl }
5565             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5566         }
5567         {
5568             \begin { pgfscope }
5569                 \@@_color_opacity: ##2
5570                 \use:c { g_@@_color _ ##1 _tl }
5571                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5572                 \pgfusepath { fill }
5573             \end { pgfscope }
5574         }
5575     }
5576     \endpgfpicture
5577 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5578 \cs_new_protected:Npn \@@_color_opacity:
5579 {
5580     \peek_meaning:NTF [
5581         { \@@_color_opacity:w }
5582         { \@@_color_opacity:w [ ] }
5583     }

```

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.

```

5584 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]

```

```

5585 {
5586   \tl_clear:N \l_tmpa_tl
5587   \keys_set_known:nn { nicematrix / color-opacity } { #1 } \l_tmpb_tl
\l_tmpa_tl (if not empty) is now the opacity and \l_tmpb_tl (if not empty) is now the colorimetric
space.
5588   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5589   \tl_if_empty:NTF \l_tmpb_tl
5590     { \@declaredcolor }
5591     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5592 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5593 \keys_define:nn { nicematrix / color-opacity }
5594 {
5595   opacity .tl_set:N          = \l_tmpa_tl ,
5596   opacity .value_required:n = true
5597 }

```

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

```

5598 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5599 {
5600   \def \l_@@_rows_tl { #1 }
5601   \def \l_@@_cols_tl { #2 }
5602   \@@_cartesian_path:
5603 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5604 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5605 {
5606   \tl_if_blank:nF { #2 }
5607   {
5608     \@@_add_to_colors_seq:en
5609     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5610     { \@@_cartesian_color:nn { #3 } { - } }
5611   }
5612 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5613 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5614 {
5615   \tl_if_blank:nF { #2 }
5616   {
5617     \@@_add_to_colors_seq:en
5618     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5619     { \@@_cartesian_color:nn { - } { #3 } }
5620   }
5621 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5622 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5623 {
5624   \tl_if_blank:nF { #2 }
5625   {
5626     \@@_add_to_colors_seq:en
5627     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5628     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5629   }
5630 }

```

The last argument is the radius of the corners of the rectangle.

```

5631 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5632 {
5633   \tl_if_blank:nF { #2 }
5634   {
5635     \@@_add_to_colors_seq:en
5636     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5637     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5638   }
5639 }

```

The last argument is the radius of the corners of the rectangle.

```

5640 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5641 {
5642   \@@_cut_on_hyphen:w #1 \q_stop
5643   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5644   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5645   \@@_cut_on_hyphen:w #2 \q_stop
5646   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5647   \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`.

```

5648   \@@_cartesian_path:n { #3 }
5649 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5650 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5651 {
5652   \clist_map_inline:nn { #3 }
5653   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5654 }

```

```

5655 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5656 {
5657   \int_step_inline:nn { \c@iRow }
5658   {
5659     \int_step_inline:nn { \c@jCol }
5660     {
5661       \int_if_even:nTF { #####1 + ##1 }
5662       { \@@_cellcolor [ #1 ] { #2 } }
5663       { \@@_cellcolor [ #1 ] { #3 } }
5664       { ##1 - #####1 }
5665     }
5666   }
5667 }

```

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”.

```

5668 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5669 {
5670   \@@_rectanglecolor [ #1 ] { #2 }
5671   { 1 - 1 }
5672   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5673 }

5674 \keys_define:nn { nicematrix / rowcolors }
5675 {
5676   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,

```



```

5677     respect-blocks .default:n = true ,
5678     cols .tl_set:N = \l_@@_cols_tl ,
5679     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5680     restart .default:n = true ,
5681     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5682 }

```

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*.

```

5683 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5684 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5685     \group_begin:
5686     \seq_clear_new:N \l_@@_colors_seq
5687     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5688     \tl_clear_new:N \l_@@_cols_tl
5689     \tl_set:Nn \l_@@_cols_tl { - }
5690     \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).

```

5691     \int_zero_new:N \l_@@_color_int
5692     \int_set_eq:NN \l_@@_color_int \c_one_int
5693     \bool_if:NT \l_@@_respect_blocks_bool
5694     {

```

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`).

```

5695         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5696         \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5697         { \@@_not_in_exterior_p:nnnnn ##1 }
5698     }
5699     \pgfpicture
5700     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5701     \clist_map_inline:nn { #2 }
5702     {
5703         \tl_set:Nn \l_tmpa_tl { ##1 }
5704         \tl_if_in:NnTF \l_tmpa_tl { - }
5705         { \@@_cut_on_hyphen:w ##1 \q_stop }
5706         { \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.

```

5707         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5708         \int_set:Nn \l_@@_color_int
5709         { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5710         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5711         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5712         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5713         \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).

```

5714     \bool_if:NT \l_@@_respect_blocks_bool
5715     {
5716         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5717         { \@@_intersect_our_row_p:nnnnn ###1 }
5718         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5719     }
5720     \tl_set:Ne \l_@@_rows_tl
5721     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5722     \tl_set:Ne \l_@@_color_tl
5723     {
5724         \@@_color_index:n
5725         {
5726             \int_mod:nn
5727             { \l_@@_color_int - 1 }
5728             { \seq_count:N \l_@@_colors_seq }
5729             + 1
5730         }
5731     }
5732     \tl_if_empty:NF \l_@@_color_tl
5733     {
5734         \@@_add_to_colors_seq:ee
5735         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5736         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5737     }
5738     \int_incr:N \l_@@_color_int
5739     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5740 }
5741 }
5742 \endpgfpicture
5743 \group_end:
5744 }

```

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.

```

5745 \cs_new:Npn \@@_color_index:n #1
5746 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5747     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5748     { \@@_color_index:n { #1 - 1 } }
5749     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5750 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5751 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5752 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5753 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5754 {
5755     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5756     { \int_set:Nn \l_tmpb_int { #3 } }
5757 }

```

```

5758 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5759 {
5760   \int_if_zero:nTF { #4 }
5761   { \prg_return_false: }
5762   {
5763     \int_compare:nNnTF { #2 } > { \c@jCol }
5764     { \prg_return_false: }
5765     { \prg_return_true: }
5766   }
5767 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5768 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5769 {
5770   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5771   { \prg_return_false: }
5772   {
5773     \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5774     { \prg_return_false: }
5775     { \prg_return_true: }
5776   }
5777 }

```

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).

```

5778 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5779 {
5780   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5781   {
5782     \bool_if:NTF \l_@@_nocolor_used_bool
5783     { \@@_cartesian_path_normal_ii: }
5784     {
5785       \clist_if_empty:NTF \l_@@_corners_cells_clist
5786       { \@@_cartesian_path_normal_i:n { #1 } }
5787       { \@@_cartesian_path_normal_ii: }
5788     }
5789   }
5790   { \@@_cartesian_path_normal_i:n { #1 } }
5791 }

```

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.

```

5792 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5793 {
5794   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5795   \clist_map_inline:Nn \l_@@_cols_tl
5796   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

5797     \def \l_tmpa_tl { ##1 }
5798     \tl_if_in:NnTF \l_tmpa_tl { - }
5799     { \@@_cut_on_hyphen:w ##1 \q_stop }
5800     { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5801     \tl_if_empty:NTF \l_tmpa_tl
5802     { \def \l_tmpa_tl { 1 } }
5803     {

```

```

5804     \str_if_eq:eeT \l_tmpa_tl { * }
5805     { \def \l_tmpa_tl { 1 } }
5806   }
5807   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5808   { \@@_error:n { Invalid~col~number } }
5809   \tl_if_empty:NTF \l_tmpb_tl
5810   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5811   {
5812     \str_if_eq:eeT \l_tmpb_tl { * }
5813     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5814   }
5815   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5816   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5817   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5818   \@@_qpoint:n { col - \l_tmpa_tl }
5819   \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5820   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5821   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5822   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5823   \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.

```

5824   \clist_map_inline:Nn \l_@@_rows_tl
5825   {
5826     \def \l_tmpa_tl { #####1 }
5827     \tl_if_in:NnTF \l_tmpa_tl { - }
5828     { \@@_cut_on_hyphen:w #####1 \q_stop }
5829     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5830     \tl_if_empty:NTF \l_tmpa_tl
5831     { \def \l_tmpa_tl { 1 } }
5832     {
5833       \str_if_eq:eeT \l_tmpa_tl { * }
5834       { \def \l_tmpa_tl { 1 } }
5835     }
5836     \tl_if_empty:NTF \l_tmpb_tl
5837     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5838     {
5839       \str_if_eq:eeT \l_tmpb_tl { * }
5840       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5841     }
5842     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5843     { \@@_error:n { Invalid~row~number } }
5844     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5845     { \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`.

```

5846   \cs_if_exist:cF
5847   { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5848   {
5849     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5850     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5851     \@@_qpoint:n { row - \l_tmpa_tl }
5852     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5853     \pgfpathrectanglecorners
5854     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5855     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5856   }
5857 }
5858 }
5859 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5860 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5861 {
5862   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5863   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5864   \clist_map_inline:Nn \l_@@_cols_tl
5865   {
5866     \@@_qpoint:n { col - ##1 }
5867     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5868     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5869     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5870     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5871     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5872     \clist_map_inline:Nn \l_@@_rows_tl
5873     {
5874       \@@_if_in_corner:nF { #####1 - ##1 }
5875       {
5876         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5877         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5878         \@@_qpoint:n { row - #####1 }
5879         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5880         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5881         {
5882           \pgfpathrectanglecorners
5883             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5884             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5885         }
5886       }
5887     }
5888   }
5889 }

```

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`).

```

5890 \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

```

5891 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5892 {
5893   \bool_set_true:N \l_@@_nocolor_used_bool
5894   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5895   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5896   \clist_map_inline:Nn \l_@@_rows_tl
5897   {
5898     \clist_map_inline:Nn \l_@@_cols_tl
5899     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5900   }
5901 }

```

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.

```

5902 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5903 {
5904   \clist_set_eq:NN \l_tmpa_clist #1

```

```

5905 \clist_clear:N #1
5906 \clist_map_inline:Nn \l_tmpa_clist
5907 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5908 \def \l_tmpa_tl { ##1 }
5909 \tl_if_in:NnTF \l_tmpa_tl { - }
5910 { \@@_cut_on_hyphen:w ##1 \q_stop }
5911 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5912 \bool_lazy_or:nnT
5913 { \str_if_eq_p:ee \l_tmpa_tl { * } }
5914 { \tl_if_blank_p:o \l_tmpa_tl }
5915 { \def \l_tmpa_tl { 1 } }
5916 \bool_lazy_or:nnT
5917 { \str_if_eq_p:ee \l_tmpb_tl { * } }
5918 { \tl_if_blank_p:o \l_tmpb_tl }
5919 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5920 \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5921 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5922 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5923 { \clist_put_right:Nn #1 { #####1 } }
5924 }
5925 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5926 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5927 {
5928 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5929 {

```

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`).

```

5930 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5931 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5932 }
5933 \ignorespaces
5934 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5935 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5936 {
5937 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5938 {
5939 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5940 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5941 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5942 }
5943 \ignorespaces
5944 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5945 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5946 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5947 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5948 {

```

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`.

```

5949 \seq_gclear:N \g_tmpa_seq
5950 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5951 { \@@_rowlistcolors_tabular:nnnn #1 }
5952 \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`.

```

5953 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5954 {
5955   { \int_use:N \c@iRow }
5956   { \exp_not:n { #1 } }
5957   { \exp_not:n { #2 } }
5958   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5959 }
5960 \ignorespaces
5961 }

```

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`).

```

5962 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5963 {
5964   \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).

```

5965 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5966 {
5967   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5968   {
5969     \@@_rowlistcolors
5970     [ \exp_not:n { #2 } ]
5971     { #1 - \int_eval:n { \c@iRow - 1 } }
5972     { \exp_not:n { #3 } }
5973     [ \exp_not:n { #4 } ]
5974   }
5975 }
5976 }

```

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.

```

5977 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5978 {
5979   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5980   { \@@_rowlistcolors_tabular_ii:nnnn #1 }
5981   \seq_gclear:N \g_@@_rowlistcolors_seq
5982 }

```

```

5983 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5984 {
5985   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5986   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5987 }

```

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.

```

5988 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5989 {

```

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).

```

5990   \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
5991   {

```

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).

```

5992   \tl_gput_left:Ne \g_@@_pre_code_before_tl
5993   {
5994     \exp_not:N \columncolor [ #1 ]
5995     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5996   }
5997 }
5998 }

```

```

5999 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6000 {
6001   \clist_map_inline:nn { #1 }
6002   {
6003     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6004     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6005     \columncolor { nocolor } { ##1 }
6006   }
6007 }
6008 \cs_new_protected:Npn \@@_EmptyRow:n #1
6009 {
6010   \clist_map_inline:nn { #1 }
6011   {
6012     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6013     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6014     \rowcolor { nocolor } { ##1 }
6015   }
6016 }

```

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`.

```
6017 \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`.

```
6018 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6019 {
6020   \int_if_zero:nTF { \l_@@_first_col_int }
6021     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6022     {
6023       \int_if_zero:nTF { \c_jCol }
6024       {
6025         \int_compare:nNnF { \c_iRow } = { -1 }
6026         {
6027           \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6028           { #1 }
6029         }
6030       }
6031       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6032     }
6033 }
```

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.

```
6034 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6035 {
6036   \int_if_zero:nF { \c_iRow }
6037   {
6038     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6039     {
6040       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6041       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6042     }
6043   }
6044 }
```

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`.

```
6045 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6046 {
6047   \IfPackageLoadedTF { tikz }
6048   {
6049     \IfPackageLoadedTF { booktabs }
6050     { #2 }
6051     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6052   }
6053   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6054 }

6055 \NewExpandableDocumentCommand { \@@_TopRule } { }
6056 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6057 \cs_new:Npn \@@_TopRule_i:
6058 {
6059   \noalign \bgroup
6060   \peek_meaning:NTF [
6061     { \@@_TopRule_ii: }
```

```

6062         { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6063     }
6064 \NewDocumentCommand \@@_TopRule_ii: { o }
6065 {
6066     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6067     {
6068         \@@_hline:n
6069         {
6070             position = \int_eval:n { \c@iRow + 1 } ,
6071             tikz =
6072             {
6073                 line~width = #1 ,
6074                 yshift = 0.25 \arrayrulewidth ,
6075                 shorten~< = - 0.5 \arrayrulewidth
6076             } ,
6077             total~width = #1
6078         }
6079     }
6080     \skip_vertical:n { \belowrulesep + #1 }
6081     \egroup
6082 }
6083 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6084 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6085 \cs_new:Npn \@@_BottomRule_i:
6086 {
6087     \noalign \bgroup
6088     \peek_meaning:NTF [
6089         { \@@_BottomRule_ii: }
6090         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6091     }
6092 \NewDocumentCommand \@@_BottomRule_ii: { o }
6093 {
6094     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6095     {
6096         \@@_hline:n
6097         {
6098             position = \int_eval:n { \c@iRow + 1 } ,
6099             tikz =
6100             {
6101                 line~width = #1 ,
6102                 yshift = 0.25 \arrayrulewidth ,
6103                 shorten~< = - 0.5 \arrayrulewidth
6104             } ,
6105             total~width = #1 ,
6106         }
6107     }
6108     \skip_vertical:N \aboverulesep
6109     \@@_create_row_node_i:
6110     \skip_vertical:n { #1 }
6111     \egroup
6112 }
6113 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6114 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6115 \cs_new:Npn \@@_MidRule_i:
6116 {
6117     \noalign \bgroup
6118     \peek_meaning:NTF [
6119         { \@@_MidRule_ii: }
6120         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6121     }
6122 \NewDocumentCommand \@@_MidRule_ii: { o }

```

```

6123 {
6124   \skip_vertical:N \aboverulesep
6125   \@@_create_row_node_i:
6126   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6127   {
6128     \@@_hline:n
6129     {
6130       position = \int_eval:n { \c@iRow + 1 } ,
6131       tikz =
6132       {
6133         line-width = #1 ,
6134         yshift = 0.25 \arrayrulewidth ,
6135         shorten~< = - 0.5 \arrayrulewidth
6136       } ,
6137       total-width = #1 ,
6138     }
6139   }
6140   \skip_vertical:n { \belowrulesep + #1 }
6141   \egroup
6142 }

```

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.

```

6143 \keys_define:nn { nicematrix / Rules }
6144 {
6145   position .int_set:N = \l_@@_position_int ,
6146   position .value_required:n = true ,
6147   start .int_set:N = \l_@@_start_int ,
6148   end .code:n =
6149     \bool_lazy_or:nnTF
6150     { \tl_if_empty_p:n { #1 } }
6151     { \str_if_eq_p:ee { #1 } { last } }
6152     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6153     { \int_set:Nn \l_@@_end_int { #1 } }
6154 }

```

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 analyse 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.

```

6155 \keys_define:nn { nicematrix / RulesBis }
6156 {
6157   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6158   multiplicity .initial:n = 1 ,
6159   dotted .bool_set:N = \l_@@_dotted_bool ,
6160   dotted .initial:n = false ,
6161   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]`.

```

6162   color .code:n =
6163     \@@_set_CTarc:n { #1 }

```

```

6164     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6165     color .value_required:n = true ,
6166     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6167     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.

```

6168     tikz .code:n =
6169     \IfPackageLoadedTF { tikz }
6170     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6171     { \@@_error:n { tikz-without-tikz } } ,
6172     tikz .value_required:n = true ,
6173     total-width .dim_set:N = \l_@@_rule_width_dim ,
6174     total-width .value_required:n = true ,
6175     width .meta:n = { total-width = #1 } ,
6176     unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6177 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6178 \cs_new_protected:Npn \@@_vline:n #1
6179 {

```

The group is for the options.

```

6180     \group_begin:
6181     \int_set_eq:NN \l_@@_end_int \c@iRow
6182     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6183     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6184     \@@_vline_i:
6185     \group_end:
6186 }
6187 \cs_new_protected:Npn \@@_vline_i:
6188 {

```

`\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`.

```

6189     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6190     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6191     \l_tmpa_tl
6192     {

```

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.

```

6193     \bool_gset_true:N \g_tmpa_bool
6194     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6195     { \@@_test_vline_in_block:nnnnn ##1 }
6196     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6197     { \@@_test_vline_in_block:nnnnn ##1 }
6198     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6199     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6200     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6201     \bool_if:NTF \g_tmpa_bool
6202     {
6203         \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.

```

6204         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6205     }
6206     {
6207         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6208         {
6209             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6210             \@@_vline_ii:
6211             \int_zero:N \l_@@_local_start_int
6212         }
6213     }
6214 }
6215 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6216 {
6217     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6218     \@@_vline_ii:
6219 }
6220 }

6221 \cs_new_protected:Npn \@@_test_in_corner_v:
6222 {
6223     \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6224     {
6225         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6226         { \bool_set_false:N \g_tmpa_bool }
6227     }
6228     {
6229         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6230         {
6231             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6232             { \bool_set_false:N \g_tmpa_bool }
6233             {
6234                 \@@_if_in_corner:nT
6235                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6236                 { \bool_set_false:N \g_tmpa_bool }
6237             }
6238         }
6239     }
6240 }

6241 \cs_new_protected:Npn \@@_vline_ii:
6242 {
6243     \tl_clear:N \l_@@_tikz_rule_tl
6244     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6245     \bool_if:NTF \l_@@_dotted_bool
6246     { \@@_vline_iv: }
6247     {
6248         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6249         { \@@_vline_iii: }
6250         { \@@_vline_v: }
6251     }
6252 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6253 \cs_new_protected:Npn \@@_vline_iii:
6254 {
6255     \pgfpicture
6256     \pgfrememberpicturepositiononpagetrue
6257     \pgf@relevantforpicturesizefalse
6258     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }

```

```

6259 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6260 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6261 \dim_set:Nn \l_tmpb_dim
6262 {
6263   \pgf@x
6264   - 0.5 \l_@@_rule_width_dim
6265   +
6266   ( \arrayrulewidth * \l_@@_multiplicity_int
6267     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6268 }
6269 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6270 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6271 \bool_lazy_all:nT
6272 {
6273   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6274   { \cs_if_exist_p:N \CT@drsc@ }
6275   { ! \tl_if_blank_p:o \CT@drsc@ }
6276 }
6277 {
6278   \group_begin:
6279   \CT@drsc@
6280   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6281   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6282   \dim_set:Nn \l_@@_tmpd_dim
6283   {
6284     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6285     * ( \l_@@_multiplicity_int - 1 )
6286   }
6287   \pgfpathrectanglecorners
6288   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6289   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6290   \pgfusepath { fill }
6291   \group_end:
6292 }
6293 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6294 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6295 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6296 {
6297   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6298   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6299   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6300   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6301 }
6302 \CT@arc@
6303 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6304 \pgfsetrectcap
6305 \pgfusepathqstroke
6306 \endpgfpicture
6307 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6308 \cs_new_protected:Npn \@@_vline_iv:
6309 {
6310   \pgfpicture
6311   \pgfrememberpicturepositiononpagetrue
6312   \pgf@relevantforpicturesizefalse
6313   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6314   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6315   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6316   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6317   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6318   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6319   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

```

6320 \CT@arc@
6321 \@@_draw_line:
6322 \endpgfpicture
6323 }

```

The following code is for the case when the user uses the key `tikz`.

```

6324 \cs_new_protected:Npn \@@_vline_v:
6325 {
6326   \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).

```

6327 \CT@arc@
6328 \tl_if_empty:NF \l_@@_rule_color_tl
6329 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6330 \pgfrememberpicturepositiononpagetrue
6331 \pgf@relevantforpicturesizefalse
6332 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6333 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6334 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6335 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6336 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6337 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6338 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6339 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6340 ( \l_tmpb_dim , \l_tmpa_dim ) --
6341 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6342 \end { tikzpicture }
6343 }

```

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).

```

6344 \cs_new_protected:Npn \@@_draw_vlines:
6345 {
6346   \int_step_inline:nnn
6347   {
6348     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6349     { 2 }
6350     { 1 }
6351   }
6352   {
6353     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6354     { \c@jCol }
6355     { \int_eval:n { \c@jCol + 1 } }
6356   }
6357   {
6358     \str_if_eq:eeF \l_@@_vlines_clist { all }
6359     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6360     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6361   }
6362 }

```

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}`.

```

6363 \cs_new_protected:Npn \@@_hline:n #1
6364 {

```

The group is for the options.

```

6365     \group_begin:
6366     \int_set_eq:NN \l_@@_end_int \c@jCol
6367     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6368     \@@_hline_i:
6369     \group_end:
6370 }

6371 \cs_new_protected:Npn \@@_hline_i:
6372 {
6373     % \int_zero:N \l_@@_local_start_int
6374     % \int_zero:N \l_@@_local_end_int

```

`\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`.

```

6375     \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6376     \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6377     \l_tmpb_tl
6378     {

```

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.

```

6379     \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6380     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6381     { \@@_test_hline_in_block:nnnnn ##1 }

6382     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6383     { \@@_test_hline_in_block:nnnnn ##1 }

6384     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6385     { \@@_test_hline_in_stroken_block:nnnn ##1 }

6386     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6387     \bool_if:NTF \g_tmpa_bool
6388     {
6389         \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.

```

6390         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6391     }
6392     {
6393         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6394         {
6395             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6396             \@@_hline_ii:
6397             \int_zero:N \l_@@_local_start_int
6398         }
6399     }
6400 }
6401 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6402 {
6403     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6404     \@@_hline_ii:
6405 }
6406 }

```

```

6407 \cs_new_protected:Npn \@@_test_in_corner_h:
6408 {
6409     \int_compare:nNnTF { \l_tmpa_tl } = { \c@iRow + 1 }
6410     {
6411         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }

```



```

6412         { \bool_set_false:N \g_tmpa_bool }
6413     }
6414     {
6415         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6416         {
6417             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6418             { \bool_set_false:N \g_tmpa_bool }
6419             {
6420                 \@@_if_in_corner:nT
6421                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6422                 { \bool_set_false:N \g_tmpa_bool }
6423             }
6424         }
6425     }
6426 }

```

```

6427 \cs_new_protected:Npn \@@_hline_ii:
6428 {
6429     \tl_clear:N \l_@@_tikz_rule_tl
6430     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6431     \bool_if:NTF \l_@@_dotted_bool
6432     { \@@_hline_iv: }
6433     {
6434         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6435         { \@@_hline_iii: }
6436         { \@@_hline_v: }
6437     }
6438 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6439 \cs_new_protected:Npn \@@_hline_iii:
6440 {
6441     \pgfpicture
6442     \pgfrememberpicturepositiononpagetrue
6443     \pgf@relevantforpicturesizefalse
6444     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6445     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6446     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6447     \dim_set:Nn \l_tmpb_dim
6448     {
6449         \pgf@y
6450         - 0.5 \l_@@_rule_width_dim
6451         +
6452         ( \arrayrulewidth * \l_@@_multiplicity_int
6453           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6454     }
6455     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6456     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6457     \bool_lazy_all:nT
6458     {
6459         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6460         { \cs_if_exist_p:N \CT@drsc@ }
6461         { ! \tl_if_blank_p:o \CT@drsc@ }
6462     }
6463     {
6464         \group_begin:
6465         \CT@drsc@
6466         \dim_set:Nn \l_@@_tmpd_dim
6467         {
6468             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6469             * ( \l_@@_multiplicity_int - 1 )
6470         }

```

```

6471     \pgfpathrectanglecorners
6472     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6473     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6474     \pgfusepathqfill
6475     \group_end:
6476 }
6477 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6478 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6479 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6480 {
6481     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6482     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6483     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6484     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6485 }
6486 \CT@arc@
6487 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6488 \pgfsetrectcap
6489 \pgfusepathqstroke
6490 \endpgfpicture
6491 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6492 \cs_new_protected:Npn \@@_hline_iv:
6493 {
6494     \pgfpicture
6495     \pgfrememberpicturepositiononpagetrue
6496     \pgf@relevantforpicturesizefalse
6497     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6498     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6499     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6500     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6501     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6502     \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6503     {
6504         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6505         \bool_if:NF \g_@@_delims_bool
6506         { \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.

```

6507     \tl_if_eq:NnF \g_@@_left_delim_tl (
6508     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6509     )
6510     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6511     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x

```

```

6512 \int_compare:nNtT { \l_@@_local_end_int } = { \c@jCol }
6513 {
6514   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6515   \bool_if:NF \g_@@_delims_bool
6516   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6517   \tl_if_eq:NnF \g_@@_right_delim_tl )
6518   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6519 }
6520 \CT@arc@
6521 \@@_draw_line:
6522 \endpgfpicture
6523 }

```

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`).

```

6524 \cs_new_protected:Npn \@@_hline_v:
6525 {
6526   \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).

```

6527   \CT@arc@
6528   \tl_if_empty:NF \l_@@_rule_color_tl
6529   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6530   \pgfrememberpicturepositiononpagetrue
6531   \pgf@relevantforpicturesizefalse
6532   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6533   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6534   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6535   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6536   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6537   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6538   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6539   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6540   ( \l_tmpa_dim , \l_tmpb_dim ) --
6541   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6542   \end { tikzpicture }
6543 }

```

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).

```

6544 \cs_new_protected:Npn \@@_draw_hlines:
6545 {
6546   \int_step_inline:nnn
6547   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6548   {
6549     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6550     { \c@iRow }
6551     { \int_eval:n { \c@iRow + 1 } } }
6552   }
6553   {
6554     \str_if_eq:eeF \l_@@_hlines_clist { all }
6555     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6556     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6557   }
6558 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6559 \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.

```

6560 \cs_set:Npn \@@_Hline_i:n #1
6561 {
6562   \peek_remove_spaces:n
6563   {
6564     \peek_meaning:NTF \Hline
6565     { \@@_Hline_ii:nn { #1 + 1 } }
6566     { \@@_Hline_iii:n { #1 } }
6567   }
6568 }
6569 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6570 \cs_set:Npn \@@_Hline_iii:n #1
6571 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6572 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6573 {
6574   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6575   \skip_vertical:N \l_@@_rule_width_dim
6576   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6577   {
6578     \@@_hline:n
6579     {
6580       multiplicity = #1 ,
6581       position = \int_eval:n { \c@iRow + 1 } ,
6582       total-width = \dim_use:N \l_@@_rule_width_dim ,
6583       #2
6584     }
6585   }
6586   \egroup
6587 }

```

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`).

```

6588 \cs_new_protected:Npn \@@_custom_line:n #1
6589 {
6590   \str_clear_new:N \l_@@_command_str
6591   \str_clear_new:N \l_@@_ccommand_str
6592   \str_clear_new:N \l_@@_letter_str
6593   \tl_clear_new:N \l_@@_other_keys_tl
6594   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

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.

```

6595   \bool_lazy_all:nTF
6596   {
6597     { \str_if_empty_p:N \l_@@_letter_str }
6598     { \str_if_empty_p:N \l_@@_command_str }
6599     { \str_if_empty_p:N \l_@@_ccommand_str }
6600   }
6601   { \@@_error:n { No~letter~and~no~command } }
6602   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6603 }
6604 \keys_define:nn { nicematrix / custom-line }
6605 {
6606   letter .str_set:N = \l_@@_letter_str ,

```

```

6607     letter .value_required:n = true ,
6608     command .str_set:N = \l_@@_command_str ,
6609     command .value_required:n = true ,
6610     ccommand .str_set:N = \l_@@_ccommand_str ,
6611     ccommand .value_required:n = true ,
6612 }

```

```

6613 \cs_new_protected:Npn \@@_custom_line_i:n #1
6614 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6615     \bool_set_false:N \l_@@_tikz_rule_bool
6616     \bool_set_false:N \l_@@_dotted_rule_bool
6617     \bool_set_false:N \l_@@_color_bool
6618     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6619     \bool_if:NT \l_@@_tikz_rule_bool
6620     {
6621         \IfPackageLoadedF { tikz }
6622         { \@@_error:n { tikz-in-custom-line-without-tikz } }
6623         \bool_if:NT \l_@@_color_bool
6624         { \@@_error:n { color-in-custom-line-with-tikz } }
6625     }
6626     \bool_if:NT \l_@@_dotted_rule_bool
6627     {
6628         \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6629         { \@@_error:n { key-multiplicity-with-dotted } }
6630     }
6631     \str_if_empty:NF \l_@@_letter_str
6632     {
6633         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6634         { \@@_error:n { Several-letters } }
6635         {
6636             \tl_if_in:NoTF
6637             \c_@@_forbidden_letters_str
6638             \l_@@_letter_str
6639             { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6640         }

```

During the analyse 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.

```

6641         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6642         { \@@_v_custom_line:n { #1 } }
6643     }
6644 }
6645 }
6646 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6647 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6648 }
6649 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6650 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6651 \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.

```

6652 \keys_define:nn { nicematrix / custom-line-bis }
6653 {
6654     multiplicity .int_set:N = \l_@@_multiplicity_int ,

```

```

6655     multiplicity .initial:n = 1 ,
6656     multiplicity .value_required:n = true ,
6657     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6658     color .value_required:n = true ,
6659     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6660     tikz .value_required:n = true ,
6661     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6662     dotted .value_forbidden:n = true ,
6663     total-width .code:n = { } ,
6664     total-width .value_required:n = true ,
6665     width .code:n = { } ,
6666     width .value_required:n = true ,
6667     sep-color .code:n = { } ,
6668     sep-color .value_required:n = true ,
6669     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6670 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6671 \bool_new:N \l_@@_dotted_rule_bool
6672 \bool_new:N \l_@@_tikz_rule_bool
6673 \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`.

```

6674 \keys_define:nn { nicematrix / custom-line-width }
6675 {
6676     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6677     multiplicity .initial:n = 1 ,
6678     multiplicity .value_required:n = true ,
6679     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6680     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6681                     \bool_set_true:N \l_@@_total_width_bool ,
6682     total-width .value_required:n = true ,
6683     width .meta:n = { total-width = #1 } ,
6684     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6685 }

```

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`).

```

6686 \cs_new_protected:Npn \@@_h_custom_line:n #1
6687 {

```

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`).

```

6688     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6689     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6690 }

```

The following command will create the command that the final user will use in its array to draw an horizontal line 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`).

```

6691 \cs_new_protected:Npn \@@_c_custom_line:n #1
6692 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6693     \exp_args:Nc \NewExpandableDocumentCommand
6694     { nicematrix - \l_@@_ccommand_str }
6695     { 0 { } m }

```

```

6696 {
6697   \noalign
6698   {
6699     \@@_compute_rule_width:n { #1 , ##1 }
6700     \skip_vertical:n { \l_@@_rule_width_dim }
6701     \clist_map_inline:nn
6702       { ##2 }
6703       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6704   }
6705 }
6706 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6707 }

```

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*.

```

6708 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6709 {
6710   \tl_if_in:nnTF { #2 } { - }
6711     { \@@_cut_on_hyphen:w #2 \q_stop }
6712     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6713   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6714   {
6715     \@@_hline:n
6716     {
6717       #1 ,
6718       start = \l_tmpa_tl ,
6719       end = \l_tmpb_tl ,
6720       position = \int_eval:n { \c@iRow + 1 } ,
6721       total-width = \dim_use:N \l_@@_rule_width_dim
6722     }
6723   }
6724 }
6725 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6726 {
6727   \bool_set_false:N \l_@@_tikz_rule_bool
6728   \bool_set_false:N \l_@@_total_width_bool
6729   \bool_set_false:N \l_@@_dotted_rule_bool
6730   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6731   \bool_if:NF \l_@@_total_width_bool
6732   {
6733     \bool_if:NTF \l_@@_dotted_rule_bool
6734       { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6735       {
6736         \bool_if:NF \l_@@_tikz_rule_bool
6737         {
6738           \dim_set:Nn \l_@@_rule_width_dim
6739             {
6740               \arrayrulewidth * \l_@@_multiplicity_int
6741               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6742             }
6743         }
6744       }
6745   }
6746 }
6747 \cs_new_protected:Npn \@@_v_custom_line:n #1
6748 {
6749   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6750   \tl_gput_right:Ne \g_@@_array_preamble_tl
6751     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6752   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6753   {

```

```

6754 \@@_vline:n
6755 {
6756   #1 ,
6757   position = \int_eval:n { \c@jCol + 1 } ,
6758   total-width = \dim_use:N \l_@@_rule_width_dim
6759 }
6760 }
6761 \@@_rec_preamble:n
6762 }
6763 \@@_custom_line:n
6764 { 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.

```

6765 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6766 {
6767   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6768   {
6769     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6770     {
6771       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6772       {
6773         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6774         { \bool_gset_false:N \g_tmpa_bool }
6775       }
6776     }
6777   }
6778 }

```

The same for vertical rules.

```

6779 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6780 {
6781   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6782   {
6783     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6784     {
6785       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6786       {
6787         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6788         { \bool_gset_false:N \g_tmpa_bool }
6789       }
6790     }
6791   }
6792 }
6793 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6794 {
6795   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6796   {
6797     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6798     {
6799       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6800       { \bool_gset_false:N \g_tmpa_bool }
6801       {
6802         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6803         { \bool_gset_false:N \g_tmpa_bool }
6804       }
6805     }
6806   }
6807 }

```



```

6808 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6809 {
6810   \int_compare:nNtT { \l_tmpa_tl } > { #1 - 1 }
6811   {
6812     \int_compare:nNtT { \l_tmpa_tl } < { #3 + 1 }
6813     {
6814       \int_compare:nNtTF { \l_tmpb_tl } = { #2 }
6815       { \bool_gset_false:N \g_tmpa_bool }
6816       {
6817         \int_compare:nNtT { \l_tmpb_tl } = { #4 + 1 }
6818         { \bool_gset_false:N \g_tmpa_bool }
6819       }
6820     }
6821   }
6822 }

```

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.

```

6823 \cs_new_protected:Npn \@@_compute_corners:
6824 {
6825   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6826   { \@@_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`).

```

6827   \clist_clear:N \l_@@_corners_cells_clist
6828   \clist_map_inline:Nn \l_@@_corners_cells_clist
6829   {
6830     \str_case:nnF { ##1 }
6831     {
6832       { NW }
6833       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6834       { NE }
6835       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6836       { SW }
6837       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6838       { SE }
6839       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6840     }
6841     { \@@_error:nn { bad~corner } { ##1 } }
6842   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6843   \clist_if_empty:NF \l_@@_corners_cells_clist
6844   {

```

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.

```

6845     \tl_gput_right:Ne \g_@@_aux_tl
6846     {
6847       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6848       { \l_@@_corners_cells_clist }
6849     }
6850   }
6851 }

```

```

6852 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6853 {
6854   \int_step_inline:nnn { #1 } { #3 }
6855   {
6856     \int_step_inline:nnn { #2 } { #4 }
6857     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6858   }
6859 }

6860 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6861 {
6862   \cs_if_exist:cTF
6863   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6864   { \prg_return_true: }
6865   { \prg_return_false: }
6866 }

```

“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.

```

6867 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6868 {

```

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.

```

6869   \bool_set_false:N \l_tmpa_bool
6870   \int_zero_new:N \l_@@_last_empty_row_int
6871   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6872   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6873   {
6874     \bool_lazy_or:nnTF
6875     {
6876       \cs_if_exist_p:c
6877       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6878     }
6879     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6880     { \bool_set_true:N \l_tmpa_bool }
6881     {
6882       \bool_if:NF \l_tmpa_bool
6883       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6884     }
6885   }

```

Now, you determine the last empty cell in the row of number 1.

```

6886   \bool_set_false:N \l_tmpa_bool
6887   \int_zero_new:N \l_@@_last_empty_column_int
6888   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6889   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6890   {
6891     \bool_lazy_or:nnTF
6892     {

```

```

6893     \cs_if_exist_p:c
6894     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6895   }
6896   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6897   { \bool_set_true:N \l_tmpa_bool }
6898   {
6899     \bool_if:NF \l_tmpa_bool
6900     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6901   }
6902 }

```

Now, we loop over the rows.

```

6903   \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6904   {

```

We treat the row number `##1` with another loop.

```

6905     \bool_set_false:N \l_tmpa_bool
6906     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6907     {
6908       \bool_lazy_or:nnTF
6909       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6910       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6911       { \bool_set_true:N \l_tmpa_bool }
6912     {
6913       \bool_if:NF \l_tmpa_bool
6914       {
6915         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6916         \clist_put_right:Nn
6917         \l_@@_corners_cells_clist
6918         { ##1 - #####1 }
6919         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6920       }
6921     }
6922   }
6923 }
6924 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6925 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6926 \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:NcT \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.

```

6927 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6928 \keys_define:nn { nicematrix / NiceMatrixBlock }
6929 {
6930   auto-columns-width .code:n =
6931   {
6932     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6933     \dim_gzero_new:N \g_@@_max_cell_width_dim
6934     \bool_set_true:N \l_@@_auto_columns_width_bool
6935   }
6936 }

```

```

6937 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6938 {
6939   \int_gincr:N \g_@@_NiceMatrixBlock_int
6940   \dim_zero:N \l_@@_columns_width_dim
6941   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6942   \bool_if:NT \l_@@_block_auto_columns_width_bool
6943   {
6944     \cs_if_exist:cT
6945     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6946     {
6947       \dim_set:Nn \l_@@_columns_width_dim
6948       {
6949         \use:c
6950         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6951       }
6952     }
6953   }
6954 }

```

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`).

```

6955 {
6956   \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.

```

6957   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6958   {
6959     \bool_if:NT \l_@@_block_auto_columns_width_bool
6960     {
6961       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6962       \iow_shipout:Ne \@mainaux
6963       {
6964         \cs_gset:cpn
6965         { @@ _ 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.

```

6966         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6967       }
6968       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6969     }
6970   }
6971   \ignorespacesafterend
6972 }

```

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).

```

6973 \cs_new_protected:Npn \@@_create_extra_nodes:
6974 {
6975   \bool_if:nTF \l_@@_medium_nodes_bool
6976   {
6977     \bool_if:NTF \l_@@_no_cell_nodes_bool
6978     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
6979     {
6980       \bool_if:NTF \l_@@_large_nodes_bool

```

```

6981         \@@_create_medium_and_large_nodes:
6982         \@@_create_medium_nodes:
6983     }
6984 }
6985 {
6986     \bool_if:NT \l_@@_large_nodes_bool
6987     {
6988         \bool_if:NTF \l_@@_no_cell_nodes_bool
6989         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6990         \@@_create_large_nodes:
6991     }
6992 }
6993 }

```

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`.

```

6994 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6995 {
6996     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6997     {
6998         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6999         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7000         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7001         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7002     }
7003     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7004     {
7005         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7006         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7007         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7008         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7009     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7010     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7011     {
7012         \int_step_variable:nnNn
7013         \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.

```

7014     {
7015         \cs_if_exist:cT
7016         { 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`.

```

7017     {
7018         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7019         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7020         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7021         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7022         {
7023             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7024             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7025         }

```

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`.

```

7026         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7027         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7028         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7029         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7030         {
7031             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7032             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7033         }
7034     }
7035 }
7036 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7037 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7038 {
7039     \dim_compare:nNnT
7040     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7041     {
7042         \@@_qpoint:n { row - \@@_i: - base }
7043         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7044         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7045     }
7046 }
7047 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7048 {
7049     \dim_compare:nNnT
7050     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7051     {
7052         \@@_qpoint:n { col - \@@_j: }
7053         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7054         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7055     }
7056 }
7057 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7058 \cs_new_protected:Npn \@@_create_medium_nodes:
7059 {
7060     \pgfpicture
7061     \pgfrememberpicturepositiononpagetrue
7062     \pgf@relevantforpicturesizefalse
7063     \@@_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”.

```

7064     \tl_set:Nn \l_@@_suffix_tl { -medium }
7065     \@@_create_nodes:

```

```

7066 \endpgfpicture
7067 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁵. 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:`.

```

7068 \cs_new_protected:Npn \@@_create_large_nodes:
7069 {
7070   \pgfpicture
7071   \pgfrememberpicturepositiononpagetrue
7072   \pgf@relevantforpicturesizefalse
7073   \@@_computations_for_medium_nodes:
7074   \@@_computations_for_large_nodes:
7075   \tl_set:Nn \l_@@_suffix_tl { - large }
7076   \@@_create_nodes:
7077   \endpgfpicture
7078 }
7079 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7080 {
7081   \pgfpicture
7082   \pgfrememberpicturepositiononpagetrue
7083   \pgf@relevantforpicturesizefalse
7084   \@@_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”.

```

7085   \tl_set:Nn \l_@@_suffix_tl { - medium }
7086   \@@_create_nodes:
7087   \@@_computations_for_large_nodes:
7088   \tl_set:Nn \l_@@_suffix_tl { - large }
7089   \@@_create_nodes:
7090   \endpgfpicture
7091 }

```

For “large nodes”, the exterior rows and columns don’t interfer. 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.

```

7092 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7093 {
7094   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7095   \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`.

```

7096   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7097   {
7098     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7099     {
7100       (
7101         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7102         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7103       )
7104       / 2
7105     }
7106     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7107     { l_@@_row _ \@@_i: _ min _ dim }
7108   }
7109   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7110 {
7111   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7112   {
7113     (
7114       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7115       \dim_use:c
7116         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7117     )
7118     / 2
7119   }
7120   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7121   { l_@@_column _ \@@_j: _ max _ dim }
7122 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7123   \dim_sub:cn
7124     { l_@@_column _ 1 _ min _ dim }
7125   \l_@@_left_margin_dim
7126   \dim_add:cn
7127     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7128     \l_@@_right_margin_dim
7129 }

```

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).

```

7130 \cs_new_protected:Npn \@@_create_nodes:
7131 {
7132   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7133   {
7134     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7135     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7136       \@@_pgf_rect_node:nnnnn
7137       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7138       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7139       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7140       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7141       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7142       \str_if_empty:NF \l_@@_name_str
7143       {
7144         \pgfnodealias
7145         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7146         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7147       }
7148     }
7149   }
7150   \int_step_inline:nn { \c@iRow }
7151   {
7152     \pgfnodealias
7153     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7154     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7155   }
7156   \int_step_inline:nn { \c@jCol }
7157   {
7158     \pgfnodealias
7159     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7160     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7161   }

```



```

7162 \pgfnodealias % added 2025-04-05
7163 { \@@_env: - last - last \l_@@_suffix_tl }
7164 { \@@_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 .

```

7165 \seq_map_pairwise_function:NNN
7166 \g_@@_multicolumn_cells_seq
7167 \g_@@_multicolumn_sizes_seq
7168 \@@_node_for_multicolumn:nn
7169 }

7170 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7171 {
7172   \cs_set_nopar:Npn \@@_i: { #1 }
7173   \cs_set_nopar:Npn \@@_j: { #2 }
7174 }

```

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”).

```

7175 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7176 {
7177   \@@_extract_coords_values: #1 \q_stop
7178   \@@_pgf_rect_node:nnnnn
7179   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7180   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7181   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7182   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7183   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7184   \str_if_empty:NF \l_@@_name_str
7185   {
7186     \pgfnodealias
7187     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7188     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7189   }
7190 }

```

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).

```

7191 \keys_define:nn { nicematrix / Block / FirstPass }
7192 {
7193   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7194           \bool_set_true:N \l_@@_p_block_bool ,
7195   j .value_forbidden:n = true ,
7196   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7197   l .value_forbidden:n = true ,
7198   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7199   r .value_forbidden:n = true ,
7200   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7201   c .value_forbidden:n = true ,

```

```

7202 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7203 L .value_forbidden:n = true ,
7204 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7205 R .value_forbidden:n = true ,
7206 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7207 C .value_forbidden:n = true ,
7208 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7209 t .value_forbidden:n = true ,
7210 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7211 T .value_forbidden:n = true ,
7212 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7213 b .value_forbidden:n = true ,
7214 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7215 B .value_forbidden:n = true ,
7216 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7217 m .value_forbidden:n = true ,
7218 v-center .meta:n = m ,
7219 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7220 p .value_forbidden:n = true ,
7221 color .code:n =
7222   \@@_color:n { #1 }
7223   \tl_set_rescan:Nnn
7224     \l_@@_draw_tl
7225     { \char_set_catcode_other:N ! }
7226     { #1 } ,
7227 color .value_required:n = true ,
7228 respect-arraystretch .code:n =
7229   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7230 respect-arraystretch .value_forbidden:n = true ,
7231 }

```

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.

```

7232 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7233 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7234 {

```

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).

```

7235   \tl_if_blank:nTF { #2 }
7236   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7237   {
7238     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7239     \@@_Block_i_czech:w \@@_Block_i:w
7240     #2 \q_stop
7241   }
7242   { #1 } { #3 } { #4 }
7243   \ignorespaces
7244 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7245 \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`.

```

7246 {
7247   \char_set_catcode_active:N -
7248   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7249 }

```

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.

```
7250 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7251 {
```

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).

```
7252 \bool_lazy_or:nnTF
7253 { \tl_if_blank_p:n { #1 } }
7254 { \str_if_eq_p:ee { * } { #1 } }
7255 { \int_set:Nn \l_tmpa_int { 100 } }
7256 { \int_set:Nn \l_tmpa_int { #1 } }
7257 \bool_lazy_or:nnTF
7258 { \tl_if_blank_p:n { #2 } }
7259 { \str_if_eq_p:ee { * } { #2 } }
7260 { \int_set:Nn \l_tmpb_int { 100 } }
7261 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7262 \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7263 {
7264   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7265   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7266   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7267 }
7268 { \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.

```
7269 \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7270 \tl_set:Ne \l_tmpa_tl
7271 {
7272   { \int_use:N \c@iRow }
7273   { \int_use:N \c@jCol }
7274   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7275   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7276 }
```

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).

```
7277 \bool_set_false:N \l_tmpa_bool
7278 \bool_if:NT \l_@@_amp_in_blocks_bool
```

\tl_if_in:nnT is slightly faster than \str_if_in:nnT.

```
7279 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7280 \bool_case:nF
7281 {
7282   \l_tmpa_bool { \@@_Block_vii:eennn }
7283   \l_@@_p_block_bool { \@@_Block_vi:eennn }
```

For the blocks mono-column, we will compose right now 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.

```

7284         \l_@@_X_bool                                { \@@_Block_v:eennn }
7285         { \tl_if_empty_p:n { #5 } }                    { \@@_Block_v:eennn }
7286         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7287         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7288     }
7289     { \@@_Block_v:eennn }
7290     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7291 }

```

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 now 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.

```

7292 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7293 {
7294     \int_gincr:N \g_@@_block_box_int
7295     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7296     {
7297         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7298         {
7299             \@@_actually_diagbox:nnnnnn
7300             { \int_use:N \c@iRow }
7301             { \int_use:N \c@jCol }
7302             { \int_eval:n { \c@iRow + #1 - 1 } }
7303             { \int_eval:n { \c@jCol + #2 - 1 } }
7304             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7305             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7306         }
7307     }
7308     \box_gclear_new:c
7309     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

7310     \hbox_gset:cn
7311     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7312     {

```

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).

```

7313         \tl_if_empty:NTF \l_@@_color_tl
7314         { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7315         { \@@_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.

```

7316         \int_compare:nNnT { #1 } = { \c_one_int }

```

```

7317         {
7318             \int_if_zero:nTF { \c@iRow }
7319             {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7320             \cs_set_eq:NN \Block \@@_NullBlock:
7321             \l_@@_code_for_first_row_tl
7322         }
7323         {
7324             \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7325             {
7326                 \cs_set_eq:NN \Block \@@_NullBlock:
7327                 \l_@@_code_for_last_row_tl
7328             }
7329         }
7330         \g_@@_row_style_tl
7331     }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7332         \@@_reset_arraystretch:
7333         \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7334         #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.).

```

7335         \@@_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}`.

```

7336         \bool_if:NTF \l_@@_tabular_bool
7337         {
7338             \bool_lazy_all:nTF
7339             {
7340                 { \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.

```

7341      {
7342      ! \dim_compare_p:nNn
7343      { \l_@@_col_width_dim } < { \c_zero_dim }
7344      }
7345      { ! \g_@@_rotate_bool }
7346      }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7347      {
7348      \use:e
7349      {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7350      \exp_not:N \begin { minipage }
7351      [ \str_lowercase:f \l_@@_vpos_block_str ]
7352      { \l_@@_col_width_dim }
7353      \str_case:on \l_@@_hpos_block_str
7354      { c \centering r \raggedleft l \raggedright }
7355      }
7356      #5
7357      \end { minipage }
7358      }

```

In the other cases, we use a `{tabular}`.

```

7359      {
7360      \bool_if:NT \c_@@_testphase_table_bool
7361      { \tagpdfsetup { table / tagging = presentation } }
7362      \use:e
7363      {
7364      \exp_not:N \begin { tabular }
7365      [ \str_lowercase:f \l_@@_vpos_block_str ]
7366      { @ { } \l_@@_hpos_block_str @ { } }
7367      }
7368      #5
7369      \end { tabular }
7370      }
7371      }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7372      {
7373      \c_math_toggle_token
7374      \use:e
7375      {
7376      \exp_not:N \begin { array }
7377      [ \str_lowercase:f \l_@@_vpos_block_str ]
7378      { @ { } \l_@@_hpos_block_str @ { } }
7379      }
7380      #5
7381      \end { array }
7382      \c_math_toggle_token
7383      }
7384      }

```

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).

```

7385      \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.

```

7386      \int_compare:nNnT { #2 } = { \c_one_int }

```

```

7387 {
7388     \dim_gset:Nn \g_@@_blocks_wd_dim
7389     {
7390         \dim_max:nn
7391         { \g_@@_blocks_wd_dim }
7392         {
7393             \box_wd:c
7394             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7395         }
7396     }
7397 }

```

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.

```

7398 \bool_lazy_and:nnT
7399 { \int_compare_p:nNn { #1 } = { \c_one_int } }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7400 { \str_if_empty_p:N \l_@@_vpos_block_str }
7401 {
7402     \dim_gset:Nn \g_@@_blocks_ht_dim
7403     {
7404         \dim_max:nn
7405         { \g_@@_blocks_ht_dim }
7406         {
7407             \box_ht:c
7408             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7409         }
7410     }
7411     \dim_gset:Nn \g_@@_blocks_dp_dim
7412     {
7413         \dim_max:nn
7414         { \g_@@_blocks_dp_dim }
7415         {
7416             \box_dp:c
7417             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7418         }
7419     }
7420 }
7421 \seq_gput_right:Ne \g_@@_blocks_seq
7422 {
7423     \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.

```

7424 {
7425     \exp_not:n { #3 } ,
7426     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7427 \bool_if:NT \g_@@_rotate_bool
7428 {
7429     \bool_if:NTF \g_@@_rotate_c_bool
7430     { m }
7431     {
7432         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7433         { T }
7434     }
7435 }
7436 }
7437 {

```

```

7438         \box_use_drop:c
7439         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7440     }
7441 }
7442 \bool_set_false:N \g_@@_rotate_c_bool
7443 }

7444 \cs_new:Npn \@@_adjust_hpos_rotate:
7445 {
7446     \bool_if:NT \g_@@_rotate_bool
7447     {
7448         \str_set:Ne \l_@@_hpos_block_str
7449         {
7450             \bool_if:NTF \g_@@_rotate_c_bool
7451             { c }
7452             {
7453                 \str_case:onF \l_@@_vpos_block_str
7454                 { b l B l t r T r }
7455                 {
7456                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7457                     { r }
7458                     { l }
7459                 }
7460             }
7461         }
7462     }
7463 }
7464 \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*.

```

7465 \cs_new_protected:Npn \@@_rotate_box_of_block:
7466 {
7467     \box_grotate:cn
7468     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7469     { 90 }
7470     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7471     {
7472         \vbox_gset_top:cn
7473         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7474         {
7475             \skip_vertical:n { 0.8 ex }
7476             \box_use:c
7477             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7478         }
7479     }
7480     \bool_if:NT \g_@@_rotate_c_bool
7481     {
7482         \hbox_gset:cn
7483         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7484         {
7485             \c_math_toggle_token
7486             \vcenter
7487             {
7488                 \box_use:c
7489                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7490             }
7491             \c_math_toggle_token
7492         }
7493     }
7494 }

```


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 now 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:nnnnnn`).

#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.

```

7495 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7496 {
7497   \seq_gput_right:Ne \g_@@_blocks_seq
7498   {
7499     \l_tmpa_tl
7500     { \exp_not:n { #3 } }
7501     {
7502       \bool_if:NTF \l_@@_tabular_bool
7503       {
7504         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7505     \@@_reset_arraystretch:
7506     \exp_not:n
7507     {
7508       \dim_zero:N \extrarowheight
7509       #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).

```

7510       \bool_if:NT \c_@@_testphase_table_bool
7511       { \tag_stop:n { table } }
7512       \use:e
7513       {
7514         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7515         { @ { } \l_@@_hpos_block_str @ { } }
7516       }
7517       #5
7518       \end { tabular }
7519     }
7520   \group_end:
7521 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7522 {
7523   \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7524     \@@_reset_arraystretch:
7525     \exp_not:n
7526     {
7527       \dim_zero:N \extrarowheight
7528       #4
7529       \c_math_toggle_token
7530       \use:e
7531       {
7532         \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7533         { @ { } \l_@@_hpos_block_str @ { } }
7534       }
7535       #5
7536       \end { array }
7537       \c_math_toggle_token
7538     }
7539   \group_end:

```

```

7540     }
7541   }
7542 }
7543 }
7544 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7545 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7546 {
7547   \seq_gput_right:Ne \g_@@_blocks_seq
7548   {
7549     \l_tmpa_tl
7550     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7551     { { \exp_not:n { #4 #5 } } }
7552   }
7553 }
7554 \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`.

```

7555 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7556 {
7557   \seq_gput_right:Ne \g_@@_blocks_seq
7558   {
7559     \l_tmpa_tl
7560     { \exp_not:n { #3 } }
7561     { \exp_not:n { #4 #5 } }
7562   }
7563 }
7564 \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).

```

7565 \keys_define:nn { nicematrix / Block / SecondPass }
7566 {
7567   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7568   ampersand-in-blocks .default:n = true ,
7569   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7570   tikz .code:n =
7571     \IfPackageLoadedTF { tikz }
7572     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7573     { \@@_error:n { tikz-key-without-tikz } } ,
7574   tikz .value_required:n = true ,
7575   fill .code:n =
7576     \tl_set_rescan:Nnn
7577     \l_@@_fill_tl
7578     { \char_set_catcode_other:N ! }
7579     { #1 } ,
7580   fill .value_required:n = true ,
7581   opacity .tl_set:N = \l_@@_opacity_tl ,
7582   opacity .value_required:n = true ,
7583   draw .code:n =
7584     \tl_set_rescan:Nnn
7585     \l_@@_draw_tl
7586     { \char_set_catcode_other:N ! }
7587     { #1 } ,
7588   draw .default:n = default ,

```

```

7589 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7590 rounded-corners .default:n = 4 pt ,
7591 color .code:n =
7592   \@@_color:n { #1 }
7593   \tl_set_rescan:Nnn
7594     \l_@@_draw_tl
7595     { \char_set_catcode_other:N ! }
7596     { #1 } ,
7597 borders .clist_set:N = \l_@@_borders_clist ,
7598 borders .value_required:n = true ,
7599 hvlines .meta:n = { vlines , hlines } ,
7600 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7601 vlines .default:n = true ,
7602 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7603 hlines .default:n = true ,
7604 line-width .dim_set:N = \l_@@_line_width_dim ,
7605 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7606 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7607   \bool_set_true:N \l_@@_p_block_bool ,
7608 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7609 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7610 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7611 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7612   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7613 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7614   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7615 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7616   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7617 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7618 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7619 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7620 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7621 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7622 m .value_forbidden:n = true ,
7623 v-center .meta:n = m ,
7624 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7625 p .value_forbidden:n = true ,
7626 name .tl_set:N = \l_@@_block_name_str ,
7627 name .value_required:n = true ,
7628 name .initial:n = ,
7629 respect-arraystretch .code:n =
7630   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7631 respect-arraystretch .value_forbidden:n = true ,
7632 transparent .bool_set:N = \l_@@_transparent_bool ,
7633 transparent .default:n = true ,
7634 transparent .initial:n = false ,
7635 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7636 }

```

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.

```

7637 \cs_new_protected:Npn \@@_draw_blocks:
7638 {
7639   \bool_if:NTF \c_@@_recent_array_bool
7640     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7641     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7642   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7643 }
7644 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7645 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```
7646 \int_zero:N \l_@@_last_row_int
7647 \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”).

```
7648 \int_compare:nNnTF { #3 } > { 98 }
7649 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7650 { \int_set:Nn \l_@@_last_row_int { #3 } }
7651 \int_compare:nNnTF { #4 } > { 98 }
7652 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7653 { \int_set:Nn \l_@@_last_col_int { #4 } }
7654 \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7655 {
7656   \bool_lazy_and:nNTF
7657   { \l_@@_preamble_bool }
7658   {
7659     \int_compare_p:n
7660     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7661   }
7662   {
7663     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7664     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7665     \@@_msg_redirect_name:nn { columns-not-used } { none }
7666   }
7667   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7668 }
7669 {
7670   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7671   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7672   {
7673     \@@_Block_v:nneenn
7674     { #1 }
7675     { #2 }
7676     { \int_use:N \l_@@_last_row_int }
7677     { \int_use:N \l_@@_last_col_int }
7678     { #5 }
7679     { #6 }
7680   }
7681 }
7682 }
```

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

```
7683 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7684 {
```

The group is for the keys.

```
7685 \group_begin:
7686 \int_compare:nNnT { #1 } = { #3 }
7687 { \str_set:Nn \l_@@_vpos_block_str { t } }
7688 \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 then `\str_if_in:nnT`.

```
7689 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
```

```

7690 \bool_lazy_and:nnT
7691   { \l_@@_vlines_block_bool }
7692   { ! \l_@@_ampersand_bool }
7693   {
7694     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7695     {
7696       \@@_vlines_block:nnn
7697       { \exp_not:n { #5 } }
7698       { #1 - #2 }
7699       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7700     }
7701   }
7702 \bool_if:NT \l_@@_hlines_block_bool
7703   {
7704     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7705     {
7706       \@@_hlines_block:nnn
7707       { \exp_not:n { #5 } }
7708       { #1 - #2 }
7709       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7710     }
7711   }
7712 \bool_if:NF \l_@@_transparent_bool
7713   {
7714     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7715     {

```

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).

```

7716     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7717     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7718   }
7719 }

7720 \tl_if_empty:NF \l_@@_draw_tl
7721   {
7722     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7723     { \@@_error:n { hlines-with-color } }
7724     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7725     {
7726       \@@_stroke_block:nnn

```

#5 are the options

```

7727       { \exp_not:n { #5 } }
7728       { #1 - #2 }
7729       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7730     }
7731     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7732     { { #1 } { #2 } { #3 } { #4 } }
7733   }

7734 \clist_if_empty:NF \l_@@_borders_clist
7735   {
7736     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7737     {
7738       \@@_stroke_borders_block:nnn
7739       { \exp_not:n { #5 } }
7740       { #1 - #2 }
7741       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7742     }
7743   }

7744 \tl_if_empty:NF \l_@@_fill_tl
7745   {

```

```

7746 \@@_add_opacity_to_fill:
7747 \tl_gput_right:Ne \g_@@_pre_code_before_tl
7748 {
7749 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7750 { #1 - #2 }
7751 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7752 { \dim_use:N \l_@@_rounded_corners_dim }
7753 }
7754 }
7755 \seq_if_empty:NF \l_@@_tikz_seq
7756 {
7757 \tl_gput_right:Ne \g_nicematrix_code_before_tl
7758 {
7759 \@@_block_tikz:nnnnn
7760 { \seq_use:Nn \l_@@_tikz_seq { , } }
7761 { #1 }
7762 { #2 }
7763 { \int_use:N \l_@@_last_row_int }
7764 { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7765 }
7766 }
7767 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7768 {
7769 \tl_gput_right:Ne \g_@@_pre_code_after_tl
7770 {
7771 \@@_actually_diagbox:nnnnnn
7772 { #1 }
7773 { #2 }
7774 { \int_use:N \l_@@_last_row_int }
7775 { \int_use:N \l_@@_last_col_int }
7776 { \exp_not:n { ##1 } }
7777 { \exp_not:n { ##2 } }
7778 }
7779 }

```

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
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7780 \pgfpicture
7781 \pgfrememberpicturepositiononpagetrue
7782 \pgf@relevantforpicturesizefalse
7783 \@@_qpoint:n { row - #1 }

```

```

7784 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7785 \@@_qpoint:n { col - #2 }
7786 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7787 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7788 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7789 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7790 \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.

```

7791 \@@_pgf_rect_node:nnnnn
7792 { \@@_env: - #1 - #2 - block }
7793 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7794 \str_if_empty:NF \l_@@_block_name_str
7795 {
7796   \pgfnodealias
7797   { \@@_env: - \l_@@_block_name_str }
7798   { \@@_env: - #1 - #2 - block }
7799   \str_if_empty:NF \l_@@_name_str
7800   {
7801     \pgfnodealias
7802     { \l_@@_name_str - \l_@@_block_name_str }
7803     { \@@_env: - #1 - #2 - block }
7804   }
7805 }

```

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.

```

7806 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7807 {
7808   \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.

```

7809 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7810 {

```

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.

```

7811 \cs_if_exist:cT
7812 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7813 {
7814   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7815   {
7816     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7817     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7818   }
7819 }
7820 }

```

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.

```

7821 \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7822 {
7823   \@@_qpoint:n { col - #2 }
7824   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7825 }
7826 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7827 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7828 {
7829   \cs_if_exist:cT

```

```

7830 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7831 {
7832   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7833   {
7834     \pgfpointanchor
7835     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7836     { east }
7837     \dim_set:Nn \l_@@_tmpd_dim
7838     { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7839   }
7840 }
7841 }
7842 \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7843 {
7844   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7845   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7846 }
7847 \@@_pgf_rect_node:nnnnn
7848 { \@@_env: - #1 - #2 - block - short }
7849 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7850 }

```

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.

```

7851 \bool_if:NT \l_@@_medium_nodes_bool
7852 {
7853   \@@_pgf_rect_node:nnn
7854   { \@@_env: - #1 - #2 - block - medium }
7855   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7856   {
7857     \pgfpointanchor
7858     { \@@_env:
7859       - \int_use:N \l_@@_last_row_int
7860       - \int_use:N \l_@@_last_col_int - medium
7861     }
7862     { south-east }
7863   }
7864 }
7865 \endpgfpicture

```

```

7866 \bool_if:NTF \l_@@_ampersand_bool
7867 {
7868   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7869   \int_zero_new:N \l_@@_split_int
7870   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7871   \pgfpicture
7872   \pgfrememberpicturepositiononpagetrue
7873   \pgf@relevantforpicturesizefalse
7874
7875   \@@_qpoint:n { row - #1 }
7876   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7877   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7878   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7879   \@@_qpoint:n { col - #2 }
7880   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7881   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7882   \dim_set:Nn \l_tmpb_dim
7883   { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7884   \bool_lazy_or:nnT
7885   { \l_@@_vlines_block_bool }
7886   { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7887   {
7888     \int_step_inline:nn { \l_@@_split_int - 1 }

```



```

7889     {
7890         \pgfpathmoveto
7891         {
7892             \pgfpoint
7893             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7894             \l_@@_tmpc_dim
7895         }
7896         \pgfpathlineto
7897         {
7898             \pgfpoint
7899             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7900             \l_@@_tmpd_dim
7901         }
7902         \CT@arc@
7903         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7904         \pgfsetrectcap
7905         \pgfusepathqstroke
7906     }
7907 }
7908 \@@_qpoint:n { row - #1 - base }
7909 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7910 \int_step_inline:nn { \l_@@_split_int }
7911 {
7912     \group_begin:
7913     \dim_set:Nn \col@sep
7914     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7915     \pgftransformshift
7916     {
7917         \pgfpoint
7918         {
7919             \l_tmpa_dim + ##1 \l_tmpb_dim -
7920             \str_case:on \l_@@_hpos_block_str
7921             {
7922                 l { \l_tmpb_dim + \col@sep }
7923                 c { 0.5 \l_tmpb_dim }
7924                 r { \col@sep }
7925             }
7926         }
7927         { \l_@@_tmpc_dim }
7928     }
7929     \pgfset { inner~sep = \c_zero_dim }
7930     \pgfnode
7931     { rectangle }
7932     {
7933         \str_case:on \l_@@_hpos_block_str
7934         {
7935             c { base }
7936             l { base~west }
7937             r { base~east }
7938         }
7939     }
7940     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7941     \group_end:
7942 }
7943 \endpgfpicture
7944 }

```

Now the case where there is no ampersand & in the content of the block.

```

7945 {
7946     \bool_if:NTF \l_@@_p_block_bool
7947     {

```

When the final user has used the key p, we have to compute the width.

```

7948     \pgfpicture

```

```

7949 \pgfrememberpicturepositiononpagetrue
7950 \pgf@relevantforpicturesizefalse
7951 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7952 {
7953   \@@_qpoint:n { col - #2 }
7954   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7955   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7956 }
7957 {
7958   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7959   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7960   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7961 }
7962 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7963 \endpgfpicture
7964 \hbox_set:Nn \l_@@_cell_box
7965 {
7966   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7967   { \g_tmpb_dim }
7968   \str_case:on \l_@@_hpos_block_str
7969   { c \centering r \raggedleft l \raggedright j { } }
7970   #6
7971   \end { minipage }
7972 }
7973 }
7974 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7975 \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.

```

7976 \pgfpicture
7977 \pgfrememberpicturepositiononpagetrue
7978 \pgf@relevantforpicturesizefalse
7979 \bool_lazy_any:nTF
7980 {
7981   { \str_if_empty_p:N \l_@@_vpos_block_str }
7982   { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7983   { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7984   { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7985 }
7986 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7987 \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`.

```

7988 \bool_if:nT \g_@@_last_col_found_bool
7989 {
7990   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
7991   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7992 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7993 \tl_set:Ne \l_tmpa_tl
7994 {
7995   \str_case:on \l_@@_vpos_block_str
7996   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7997   { } {
7998     \str_case:on \l_@@_hpos_block_str
7999     {

```

```

8000             c { center }
8001             l { west }
8002             r { east }
8003             j { center }
8004         }
8005     }
8006     c {
8007         \str_case:on \l_@@_hpos_block_str
8008         {
8009             c { center }
8010             l { west }
8011             r { east }
8012             j { center }
8013         }
8014
8015     }
8016     T {
8017         \str_case:on \l_@@_hpos_block_str
8018         {
8019             c { north }
8020             l { north~west }
8021             r { north~east }
8022             j { north }
8023         }
8024
8025     }
8026     B {
8027         \str_case:on \l_@@_hpos_block_str
8028         {
8029             c { south }
8030             l { south~west }
8031             r { south~east }
8032             j { south }
8033         }
8034
8035     }
8036 }
8037
8038 \pgftransformshift
8039 {
8040     \pgfpointanchor
8041     {
8042         \@@_env: - #1 - #2 - block
8043         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8044     }
8045     { \l_tmpa_tl }
8046 }
8047 \pgfset { inner~sep = \c_zero_dim }
8048 \pgfnode
8049 { rectangle }
8050 { \l_tmpa_tl }
8051 { \box_use_drop:N \l_@@_cell_box } { } { }
8052 }

```

End of the case when $\backslash l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

8053 {
8054     \pgfextracty \l_tmpa_dim
8055     {
8056         \@@_qpoint:n
8057         {
8058             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8059             - base
8060         }

```

```

8061     }
8062     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8063     \pgfpointanchor
8064     {
8065         \@@_env: - #1 - #2 - block
8066         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8067     }
8068     {
8069         \str_case:on \l_@@_hpos_block_str
8070         {
8071             c { center }
8072             l { west }
8073             r { east }
8074             j { center }
8075         }
8076     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8077     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8078     \pgfset { inner~sep = \c_zero_dim }
8079     \pgfnode
8080     { rectangle }
8081     {
8082         \str_case:on \l_@@_hpos_block_str
8083         {
8084             c { base }
8085             l { base~west }
8086             r { base~east }
8087             j { base }
8088         }
8089     }
8090     { \box_use_drop:N \l_@@_cell_box } { } { }
8091 }
8092 \endpgfpicture
8093 }
8094 \group_end:
8095 }
8096 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8097 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8098 {
8099     \pgfpicture
8100     \pgfrememberpicturepositiononpagetrue
8101     \pgf@relevantforpicturesizefalse
8102     \pgfpathrectanglecorners
8103     { \pgfpoint { #2 } { #3 } }
8104     { \pgfpoint { #4 } { #5 } }
8105     \pgfsetfillcolor { #1 }
8106     \pgfusepath { fill }
8107     \endpgfpicture
8108 }

```

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).

```

8109 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8110 {
8111     \tl_if_empty:NF \l_@@_opacity_tl

```

```

8112 {
8113   \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8114     {
8115       \tl_set:Ne \l_@@_fill_tl
8116       {
8117         [ opacity = \l_@@_opacity_tl ,
8118           \tl_tail:o \l_@@_fill_tl
8119       ]
8120     }
8121     {
8122       \tl_set:Ne \l_@@_fill_tl
8123       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8124     }
8125   ]
8126 }

```

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).

```

8127 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8128 {
8129   \group_begin:
8130   \tl_clear:N \l_@@_draw_tl
8131   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8132   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8133   \pgfpicture
8134   \pgfrememberpicturepositiononpagetrue
8135   \pgf@relevantforpicturesizefalse
8136   \tl_if_empty:NF \l_@@_draw_tl
8137   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8138     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8139     { \CT@arc@ }
8140     { \@@_color:o \l_@@_draw_tl }
8141   }
8142   \pgfsetcornersarced
8143   {
8144     \pgfpoint
8145     { \l_@@_rounded_corners_dim }
8146     { \l_@@_rounded_corners_dim }
8147   }
8148   \@@_cut_on_hyphen:w #2 \q_stop
8149   \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8150   {
8151     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8152     {
8153       \@@_qpoint:n { row - \l_tmpa_tl }
8154       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8155       \@@_qpoint:n { col - \l_tmpb_tl }
8156       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8157       \@@_cut_on_hyphen:w #3 \q_stop
8158       \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8159       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8160       \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8161       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8162       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8163       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8164       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8165       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8166       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8167       \pgfpathrectanglecorners

```

```

8168         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8169         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8170         \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8171         { \pgfusepathqstroke }
8172         { \pgfusepath { stroke } }
8173     }
8174 }
8175 \endpgfpicture
8176 \group_end:
8177 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8178 \keys_define:nn { nicematrix / BlockStroke }
8179 {
8180     color .tl_set:N = \l_@@_draw_tl ,
8181     draw .code:n =
8182         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8183     draw .default:n = default ,
8184     line-width .dim_set:N = \l_@@_line_width_dim ,
8185     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8186     rounded-corners .default:n = 4 pt
8187 }

```

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).

```

8188 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8189 {
8190     \group_begin:
8191     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8192     \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8193     \@@_cut_on_hyphen:w #2 \q_stop
8194     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8195     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8196     \@@_cut_on_hyphen:w #3 \q_stop
8197     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8198     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8199     \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8200     {
8201         \use:e
8202         {
8203             \@@_vline:n
8204             {
8205                 position = ##1 ,
8206                 start = \l_@@_tmpc_tl ,
8207                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
8208                 total-width = \dim_use:N \l_@@_line_width_dim
8209             }
8210         }
8211     }
8212     \group_end:
8213 }
8214 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8215 {
8216     \group_begin:
8217     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8218     \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8219     \@@_cut_on_hyphen:w #2 \q_stop
8220     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8221     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8222     \@@_cut_on_hyphen:w #3 \q_stop
8223     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8224     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }

```

```

8225 \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8226 {
8227   \use:e
8228   {
8229     \@@_hline:n
8230     {
8231       position = ##1 ,
8232       start = \l_@@_tmpd_tl ,
8233       end = \int_eval:n { \l_tmpb_tl - 1 } ,
8234       total-width = \dim_use:N \l_@@_line_width_dim
8235     }
8236   }
8237 }
8238 \group_end:
8239 }

```

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).

```

8240 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8241 {
8242   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8243   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8244   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8245   { \@@_error:n { borders~forbidden } }
8246   {
8247     \tl_clear_new:N \l_@@_borders_tikz_tl
8248     \keys_set:no
8249     { nicematrix / OnlyForTikzInBorders }
8250     \l_@@_borders_clist
8251     \@@_cut_on_hyphen:w #2 \q_stop
8252     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8253     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8254     \@@_cut_on_hyphen:w #3 \q_stop
8255     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8256     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8257     \@@_stroke_borders_block_i:
8258   }
8259 }
8260 \hook_gput_code:nnn { begindocument } { . }
8261 {
8262   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8263   {
8264     \c_@@_pgfortikzpicture_tl
8265     \@@_stroke_borders_block_ii:
8266     \c_@@_endpgfortikzpicture_tl
8267   }
8268 }
8269 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8270 {
8271   \pgfrememberpicturepositiononpagetrue
8272   \pgf@relevantforpicturesizefalse
8273   \CT@arc@
8274   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8275   \clist_if_in:NnT \l_@@_borders_clist { right }
8276   { \@@_stroke_vertical:n \l_tmpb_tl }
8277   \clist_if_in:NnT \l_@@_borders_clist { left }
8278   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8279   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8280   { \@@_stroke_horizontal:n \l_tmpa_tl }
8281   \clist_if_in:NnT \l_@@_borders_clist { top }
8282   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }

```

```

8283 }
8284 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8285 {
8286   tikz .code:n =
8287     \cs_if_exist:NTF \tikzpicture
8288     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8289     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8290   tikz .value_required:n = true ,
8291   top .code:n = ,
8292   bottom .code:n = ,
8293   left .code:n = ,
8294   right .code:n = ,
8295   unknown .code:n = \@@_error:n { bad~border }
8296 }

```

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).

```

8297 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8298 {
8299   \@@_qpoint:n \l_@@_tmpc_tl
8300   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8301   \@@_qpoint:n \l_tmpa_tl
8302   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8303   \@@_qpoint:n { #1 }
8304   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8305   {
8306     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8307     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8308     \pgfusepathqstroke
8309   }
8310   {
8311     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8312     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8313   }
8314 }

```

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).

```

8315 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8316 {
8317   \@@_qpoint:n \l_@@_tmpd_tl
8318   \clist_if_in:NnTF \l_@@_borders_clist { left }
8319   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8320   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8321   \@@_qpoint:n \l_tmpb_tl
8322   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8323   \@@_qpoint:n { #1 }
8324   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8325   {
8326     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8327     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8328     \pgfusepathqstroke
8329   }
8330   {
8331     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8332     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8333   }
8334 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8335 \keys_define:nn { nicematrix / BlockBorders }
8336 {

```



```

8337     borders .clist_set:N = \l_@@_borders_clist ,
8338     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8339     rounded-corners .default:n = 4 pt ,
8340     line-width .dim_set:N = \l_@@_line_width_dim
8341 }

```

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.

```

8342 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8343 {
8344     \begin { tikzpicture }
8345     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8346     \clist_map_inline:nn { #1 }
8347     {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8348         \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8349         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8350         (
8351             [
8352                 xshift = \dim_use:N \l_@@_offset_dim ,
8353                 yshift = - \dim_use:N \l_@@_offset_dim
8354             ]
8355             #2 -| #3
8356         )
8357         rectangle
8358         (
8359             [
8360                 xshift = - \dim_use:N \l_@@_offset_dim ,
8361                 yshift = \dim_use:N \l_@@_offset_dim
8362             ]
8363             \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8364         ) ;
8365     }
8366     \end { tikzpicture }
8367 }
8368 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8369 \keys_define:nn { nicematrix / SpecialOffset }
8370 { 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.

```

8371 \cs_new_protected:Npn \@@_NullBlock:
8372 { \@@_collect_options:n { \@@_NullBlock_i: } }
8373 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8374 { }

```

27 How to draw the dotted lines transparently

```

8375 \cs_set_protected:Npn \@@_renew_matrix:
8376 {
8377   \RenewDocumentEnvironment { pmatrix } { }
8378     { \pNiceMatrix }
8379     { \endpNiceMatrix }
8380   \RenewDocumentEnvironment { vmatrix } { }
8381     { \vNiceMatrix }
8382     { \endvNiceMatrix }
8383   \RenewDocumentEnvironment { Vmatrix } { }
8384     { \VNiceMatrix }
8385     { \endVNiceMatrix }
8386   \RenewDocumentEnvironment { bmatrix } { }
8387     { \bNiceMatrix }
8388     { \endbNiceMatrix }
8389   \RenewDocumentEnvironment { Bmatrix } { }
8390     { \BNiceMatrix }
8391     { \endBNiceMatrix }
8392 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8393 \keys_define:nn { nicematrix / Auto }
8394 {
8395   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8396   columns-type .value_required:n = true ,
8397   l .meta:n = { columns-type = l } ,
8398   r .meta:n = { columns-type = r } ,
8399   c .meta:n = { columns-type = c } ,
8400   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8401   delimiters / color .value_required:n = true ,
8402   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8403   delimiters / max-width .default:n = true ,
8404   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8405   delimiters .value_required:n = true ,
8406   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8407   rounded-corners .default:n = 4 pt
8408 }
8409 \NewDocumentCommand \AutoNiceMatrixWithDelims
8410 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8411 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8412 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8413 {

```

The group is for the protection of the keys.

```

8414 \group_begin:
8415 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8416 \use:e
8417 {
8418   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8419     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8420     [ \exp_not:o \l_tmpa_tl ]
8421 }
8422 \int_if_zero:nT { \l_@@_first_row_int }
8423 {
8424   \int_if_zero:nT { \l_@@_first_col_int } { & }
8425   \prg_replicate:nn { #4 - 1 } { & }
8426   \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\

```

```

8427     }
8428     \prg_replicate:nn { #3 }
8429     {
8430         \int_if_zero:nT { \l_@@_first_col_int } { & }
8431         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8432         \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8433     }
8434     \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8435     {
8436         \int_if_zero:nT { \l_@@_first_col_int } { & }
8437         \prg_replicate:nn { #4 - 1 } { & }
8438         \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8439     }
8440     \end { NiceArrayWithDelims }
8441     \group_end:
8442 }

8443 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8444 {
8445     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8446     {
8447         \bool_gset_true:N \g_@@_delims_bool
8448         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8449         \AutoNiceMatrixWithDelims { #2 } { #3 }
8450     }
8451 }

8452 \@@_define_com:NNN p ( )
8453 \@@_define_com:NNN b [ ]
8454 \@@_define_com:NNN v | |
8455 \@@_define_com:NNN V \l \l
8456 \@@_define_com:NNN B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8457 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8458 {
8459     \group_begin:
8460     \bool_gset_false:N \g_@@_delims_bool
8461     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8462     \group_end:
8463 }

```

29 The redefinition of the command `\dotfill`

```

8464 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8465 \cs_new_protected:Npn \@@_dotfill:
8466 {

```

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}`).

```

8467     \@@_old_dotfill:
8468     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8469 }

```

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`.

```

8470 \cs_new_protected:Npn \@@_dotfill_i:
8471 {
8472   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8473   { \@@_old_dotfill: }
8474 }

```

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.

```

8475 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8476 {
8477   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8478   {
8479     \@@_actually_diagbox:nnnnnn
8480     { \int_use:N \c@iRow }
8481     { \int_use:N \c@jCol }
8482     { \int_use:N \c@iRow }
8483     { \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.

```

8484     { \g_@@_row_style_tl \exp_not:n { #1 } }
8485     { \g_@@_row_style_tl \exp_not:n { #2 } }
8486   }

```

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`.

```

8487   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8488   {
8489     { \int_use:N \c@iRow }
8490     { \int_use:N \c@jCol }
8491     { \int_use:N \c@iRow }
8492     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8493     { }
8494   }
8495 }

```

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.

```

8496 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8497 {
8498   \pgfpicture
8499   \pgf@relevantforpicturesizefalse
8500   \pgfrememberpicturepositiononpagetrue
8501   \@@_qpoint:n { row - #1 }
8502   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8503   \@@_qpoint:n { col - #2 }
8504   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8505   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8506   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8507   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8508   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8509   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

```

8510 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8511 {

```

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.

```

8512 \CT@arc@
8513 \pgfsetroundcap
8514 \pgfusepathqstroke
8515 }
8516 \pgfset { inner~sep = 1 pt }
8517 \pgfscope
8518 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8519 \pgfnode { rectangle } { south~west }
8520 {
8521 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8522 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8523 \end { minipage }
8524 }
8525 { }
8526 { }
8527 \endpgfscope
8528 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8529 \pgfnode { rectangle } { north~east }
8530 {
8531 \begin { minipage } { 20 cm }
8532 \raggedleft
8533 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8534 \end { minipage }
8535 }
8536 { }
8537 { }
8538 \endpgfpicture
8539 }

```

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. 85.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8540 \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 `\`.

```

8541 \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`.

```

8542 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8543 {
8544 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8545 \@@_CodeAfter_iv:n
8546 }

```

We catch the argument of the command `\end` (in `#1`).

```
8547 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8548 {
```

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.

```
8549 \str_if_eq:eeTF \currenvir { #1 }
8550 { \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`.

```
8551 {
8552 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8553 \@@_CodeAfter_ii:n
8554 }
8555 }
```

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.

```
8556 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8557 {
8558 \pgfpicture
8559 \pgfrememberpicturepositiononpagetrue
8560 \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.

```
8561 \@@_qpoint:n { row - 1 }
8562 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8563 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8564 \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).

```
8565 \bool_if:nTF { #3 }
8566 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8567 { \dim_set:NN \l_tmpa_dim { - \c_max_dim } }
8568 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8569 {
8570 \cs_if_exist:cT
8571 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8572 {
8573 \pgfpointanchor
8574 { \@@_env: - ##1 - #2 }
8575 { \bool_if:nTF { #3 } { west } { east } }
8576 \dim_set:NN \l_tmpa_dim
8577 {
8578 \bool_if:nTF { #3 }
8579 { \dim_min:nn }
```

```

8580         { \dim_max:nn }
8581         \l_tmpa_dim
8582         { \pgf@x }
8583     }
8584 }
8585 }

```

Now we can put the delimiter with a node of PGF.

```

8586 \pgfset { inner~sep = \c_zero_dim }
8587 \dim_zero:N \nulldelimiterspace
8588 \pgftransformshift
8589 {
8590     \pgfpoint
8591     { \l_tmpa_dim }
8592     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8593 }
8594 \pgfnode
8595 { rectangle }
8596 { \bool_if:nTF { #3 } { east } { west } }
8597 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8598     \nullfont
8599     \c_math_toggle_token
8600     \@@_color:o \l_@@_delimiters_color_tl
8601     \bool_if:nTF { #3 } { \left #1 } { \left . }
8602     \vcenter
8603     {
8604         \nullfont
8605         \hrule \@height
8606             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8607             \@depth \c_zero_dim
8608             \@width \c_zero_dim
8609     }
8610     \bool_if:nTF { #3 } { \right . } { \right #1 }
8611     \c_math_toggle_token
8612 }
8613 { }
8614 { }
8615 \endpgfpicture
8616 }

```

33 The command \SubMatrix

```

8617 \keys_define:nn { nicematrix / sub-matrix }
8618 {
8619     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8620     extra-height .value_required:n = true ,
8621     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8622     left-xshift .value_required:n = true ,
8623     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8624     right-xshift .value_required:n = true ,
8625     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8626     xshift .value_required:n = true ,
8627     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8628     delimiters / color .value_required:n = true ,
8629     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8630     slim .default:n = true ,
8631     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8632     hlines .default:n = all ,
8633     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,

```

```

8634     vlines .default:n = all ,
8635     hvlines .meta:n = { hlines, vlines } ,
8636     hvlines .value_forbidden:n = true
8637   }
8638   \keys_define:nn { nicematrix }
8639   {
8640     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8641     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8642     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8643     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8644   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8645 \keys_define:nn { nicematrix / SubMatrix }
8646 {
8647   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8648   delimiters / color .value_required:n = true ,
8649   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8650   hlines .default:n = all ,
8651   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8652   vlines .default:n = all ,
8653   hvlines .meta:n = { hlines, vlines } ,
8654   hvlines .value_forbidden:n = true ,
8655   name .code:n =
8656     \tl_if_empty:nTF { #1 }
8657     { \@@_error:n { Invalid-name } }
8658     {
8659       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8660       {
8661         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8662         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8663         {
8664           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8665           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8666         }
8667       }
8668       { \@@_error:n { Invalid-name } }
8669     } ,
8670   name .value_required:n = true ,
8671   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8672   rules .value_required:n = true ,
8673   code .tl_set:N = \l_@@_code_tl ,
8674   code .value_required:n = true ,
8675   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8676 }

8677 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8678 {
8679   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8680   {
8681     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8682     [
8683       delimiters / color = \l_@@_delimiters_color_tl ,
8684       hlines = \l_@@_submatrix_hlines_clist ,
8685       vlines = \l_@@_submatrix_vlines_clist ,
8686       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8687       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8688       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8689       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8690       #5
8691     ]
8692   }

```



```

8693 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8694 \ignorespaces
8695 }
8696 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8697 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8698 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8699 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8700 {
8701 \seq_gput_right:Ne \g_@@_submatrix_seq
8702 {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8703 { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8704 { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8705 { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8706 { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8707 }
8708 }

```

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`.

```

8709 \hook_gput_code:nnn { begindocument } { . }
8710 {
8711 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m 0 { } E { _ ^ } { { } { } } }
8712 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8713 { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8714 }
8715 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8716 {
8717 \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8718 \@@_compute_i_j:nn { #2 } { #3 }
8719 \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8720 { \def \arraystretch { 1 } }
8721 \bool_lazy_or:nnTF
8722 { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8723 { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8724 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8725 {
8726 \str_clear_new:N \l_@@_submatrix_name_str
8727 \keys_set:nn { nicematrix / SubMatrix } { #5 }
8728 \pgfpicture
8729 \pgfrememberpicturepositiononpagetrue
8730 \pgf@relevantforpicturesizefalse
8731 \pgfset { inner~sep = \c_zero_dim }
8732 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8733 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currifcation.

```

8734 \bool_if:NTF \l_@@_submatrix_slim_bool
8735 { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8736 { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8737 {
8738   \cs_if_exist:cT
8739   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8740   {
8741     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8742     \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
8743     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8744   }
8745   \cs_if_exist:cT
8746   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8747   {
8748     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8749     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
8750     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8751   }
8752 }
8753 \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
8754 { \@@_error:nn { Impossible-delimiter } { left } }
8755 {
8756   \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }
8757   { \@@_error:nn { Impossible-delimiter } { right } }
8758   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8759 }
8760 \endpgfpicture
8761 }
8762 \group_end:
8763 \ignorespaces
8764 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8765 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8766 {
8767   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8768   \dim_set:Nn \l_@@_y_initial_dim
8769   {
8770     \fp_to_dim:n
8771     {
8772       \pgf@y
8773       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8774     }
8775   }
8776   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8777   \dim_set:Nn \l_@@_y_final_dim
8778   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8779   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8780   {
8781     \cs_if_exist:cT
8782     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8783     {
8784       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8785       \dim_set:Nn \l_@@_y_initial_dim
8786       { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8787     }
8788     \cs_if_exist:cT
8789     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8790     {
8791       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8792       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8793       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }

```

```

8794     }
8795   }
8796   \dim_set:Nn \l_tmpa_dim
8797   {
8798     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8799     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8800   }
8801   \dim_zero:N \nullldelimiterspace

```

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).

```

8802 \NewDocumentCommand \@@_compute_i_j:nn
8803 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8804 { \@@_compute_i_j:nnnn #1 #2 }

8805 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8806 {
8807   \def \l_@@_first_i_tl { #1 }
8808   \def \l_@@_first_j_tl { #2 }
8809   \def \l_@@_last_i_tl { #3 }
8810   \def \l_@@_last_j_tl { #4 }
8811   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8812   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8813   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8814   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8815   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8816   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8817   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8818   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8819 }

```

We will draw the rules in the `\SubMatrix`.

```

8820 \group_begin:
8821 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8822 \@@_set_CTarc:o \l_@@_rules_color_tl
8823 \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`.

```

8824 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8825 {
8826   \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8827   {
8828     \int_compare:nNnT
8829     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8830     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8831       \@@_qpoint:n { col - ##1 }
8832       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8833       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8834       \pgfusepathqstroke
8835     }
8836   }
8837 }

```

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.

```

8838 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8839 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }

```

```

8840 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8841 {
8842   \bool_lazy_and:nnTF
8843   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8844   {
8845     \int_compare_p:nNn
8846     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8847   {
8848     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8849     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8850     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8851     \pgfusepathqstroke
8852   }
8853   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8854 }

```

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.

```

8855 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8856 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8857 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8858 {
8859   \bool_lazy_and:nnTF
8860   { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8861   {
8862     \int_compare_p:nNn
8863     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8864   {
8865     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8866   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8867   \dim_set:Nn \l_tmpa_dim
8868   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8869   \str_case:nn { #1 }
8870   {
8871     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8872     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8873     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8874   }
8875   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8876   \dim_set:Nn \l_tmpb_dim
8877   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8878   \str_case:nn { #2 }
8879   {
8880     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8881     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8882     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8883   }
8884   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8885   \pgfusepathqstroke
8886   \group_end:
8887 }
8888 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8889 }

```

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).

```

8890 \str_if_empty:NF \l_@@_submatrix_name_str
8891 {

```

```

8892      \l_@@_pgf_rect_node:nnnn \l_@@_submatrix_name_str
8893      \l_@@_x_initial_dim \l_@@_y_initial_dim
8894      \l_@@_x_final_dim \l_@@_y_final_dim
8895    }
8896  \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`.

```

8897  \begin { pgfscope }
8898  \pgftransformshift
8899  {
8900    \pgfpoint
8901    { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8902    { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8903  }
8904  \str_if_empty:NTF \l_@@_submatrix_name_str
8905  { \@@_node_left:nn #1 { } }
8906  { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8907  \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8908  \pgftransformshift
8909  {
8910    \pgfpoint
8911    { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8912    { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8913  }
8914  \str_if_empty:NTF \l_@@_submatrix_name_str
8915  { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8916  {
8917    \@@_node_right:nnnn #2
8918    { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8919  }

```

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`.

```

8920  \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8921  \flag_clear_new:N \l_@@_code_flag
8922  \l_@@_code_tl
8923  }

```

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`.

```

8924  \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 currying.

```

8925  \cs_new:Npn \@@_pgfpointanchor:n #1
8926  { \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`).

```

8927 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8928 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
8929 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8930 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8931 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

8932 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

8933 { \@@_pgfpointanchor_ii:n { #1 } }
8934 }

```

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`).

```

8935 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8936 { \@@_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`.

```

8937 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

8938 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8939 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8940 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

8941 { \@@_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`).

```

8942 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8943 }

```

The following function is for the case when the name contains an hyphen.

```

8944 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8945 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8946 \@@_env:
8947 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8948 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8949 }

```

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.

```

8950 \tl_const:Nn \c_@@_integers_alist_tl
8951 {
8952 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8953 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8954 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8955 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8956 }

```

```

8957 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8958 {

```

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`.

```

8959 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8960 {
8961 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8962 \@@_env: -
8963 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8964 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8965 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8966 }
8967 {
8968 \str_if_eq:eeTF { #1 } { last }
8969 {
8970 \flag_raise:N \l_@@_code_flag
8971 \@@_env: -
8972 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8973 { \int_eval:n { \l_@@_last_i_tl + 1 } }
8974 { \int_eval:n { \l_@@_last_j_tl + 1 } }
8975 }
8976 { #1 }
8977 }
8978 }

```

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`).

```

8979 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8980 {
8981 \pgfnode
8982 { rectangle }
8983 { east }
8984 {
8985 \nullfont
8986 \c_math_toggle_token
8987 \@@_color:o \l_@@_delimiters_color_tl
8988 \left #1
8989 \vcenter
8990 {
8991 \nullfont
8992 \hrule \@height \l_tmpa_dim
8993 \@depth \c_zero_dim
8994 \@width \c_zero_dim
8995 }
8996 \right .
8997 \c_math_toggle_token
8998 }
8999 { #2 }
9000 { }
9001 }

```

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.

```

9002 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9003 {
9004   \pgfnode
9005     { rectangle }
9006     { west }
9007     {
9008       \nullfont
9009       \c_math_toggle_token
9010       \colorlet { current-color } { . }
9011       \@@_color:o \l_@@_delimiters_color_tl
9012       \left .
9013       \vcenter
9014         {
9015           \nullfont
9016           \hrule \@height \l_tmpa_dim
9017             \@depth \c_zero_dim
9018             \@width \c_zero_dim
9019         }
9020       \right #1
9021       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9022       ^ { \color { current-color } \smash { #4 } }
9023       \c_math_toggle_token
9024     }
9025     { #2 }
9026     { }
9027 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9028 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9029 {
9030   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9031   \ignorespaces
9032 }
9033 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9034 {
9035   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9036   \ignorespaces
9037 }
9038 \keys_define:nn { nicematrix / Brace }
9039 {
9040   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9041   left-shorten .default:n = true ,
9042   left-shorten .value_forbidden:n = true ,
9043   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9044   right-shorten .default:n = true ,
9045   right-shorten .value_forbidden:n = true ,
9046   shorten .meta:n = { left-shorten , right-shorten } ,
9047   shorten .value_forbidden:n = true ,
9048   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9049   yshift .value_required:n = true ,
9050   yshift .initial:n = \c_zero_dim ,
9051   color .tl_set:N = \l_tmpa_tl ,

```



```

9052     color .value_required:n = true ,
9053     unknown .code:n = \@_error:n { Unknown~key~for~Brace }
9054 }

```

#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*.

```

9055 \cs_new_protected:Npn \@_brace:nnnnn #1 #2 #3 #4 #5
9056 {
9057   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9058   \@_compute_i_j:nn { #1 } { #2 }
9059   \bool_lazy_or:nnTF
9060     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9061     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9062   {
9063     \str_if_eq:eeTF { #5 } { under }
9064     { \@_error:nn { Construct~too~large } { \UnderBrace } }
9065     { \@_error:nn { Construct~too~large } { \OverBrace } }
9066   }
9067   {
9068     \tl_clear:N \l_tmpa_tl
9069     \keys_set:nn { nicematrix / Brace } { #4 }
9070     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9071     \pgfpicture
9072     \pgfrememberpicturepositiononpagetrue
9073     \pgf@relevantforpicturesizefalse
9074     \bool_if:NT \l_@@_brace_left_shorten_bool
9075     {
9076       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9077       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9078       {
9079         \cs_if_exist:cT
9080           { pgf @ sh @ ns @ \@_env: - ##1 - \l_@@_first_j_tl }
9081           {
9082             \pgfpointanchor { \@_env: - ##1 - \l_@@_first_j_tl } { west }
9083
9084             \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9085             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9086           }
9087       }
9088     }
9089     \bool_lazy_or:nnT
9090       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9091       { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9092     {
9093       \@_qpoint:n { col - \l_@@_first_j_tl }
9094       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9095     }
9096     \bool_if:NT \l_@@_brace_right_shorten_bool
9097     {
9098       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9099       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
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     }

```

```

9110 \bool_lazy_or:nnT
9111 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9112 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9113 {
9114   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9115   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9116 }
9117 \pgfset { inner~sep = \c_zero_dim }
9118 \str_if_eq:eeTF { #5 } { under }
9119 { \@@_underbrace_i:n { #3 } }
9120 { \@@_overbrace_i:n { #3 } }
9121 \endpgfpicture
9122 }
9123 \group_end:
9124 }

```

The argument is the text to put above the brace.

```

9125 \cs_new_protected:Npn \@@_overbrace_i:n #1
9126 {
9127   \@@_qpoint:n { row - \l_@@_first_i_tl }
9128   \pgftransformshift
9129   {
9130     \pgfpoint
9131     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9132     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9133   }
9134   \pgfnode
9135   { rectangle }
9136   { south }
9137   {
9138     \vtop
9139     {
9140       \group_begin:
9141       \everycr { }
9142       \halign
9143       {
9144         \hfil ## \hfil \crcr
9145         \bool_if:NTF \l_@@_tabular_bool
9146         { \begin { tabular } { c } #1 \end { tabular } }
9147         { $ \begin { array } { c } #1 \end { array } $ }
9148         \cr
9149         \c_math_toggle_token
9150         \overbrace
9151         {
9152           \hbox_to_wd:nn
9153           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9154           { }
9155         }
9156         \c_math_toggle_token
9157         \cr
9158       }
9159       \group_end:
9160     }
9161   }
9162   { }
9163   { }
9164 }

```

The argument is the text to put under the brace.

```

9165 \cs_new_protected:Npn \@@_underbrace_i:n #1
9166 {
9167   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9168   \pgftransformshift

```

```

9169 {
9170   \pgfpoint
9171   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9172   { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9173 }
9174 \pgfnode
9175 { rectangle }
9176 { north }
9177 {
9178   \group_begin:
9179   \everycr { }
9180   \vbox
9181   {
9182     \halign
9183     {
9184       \hfil ## \hfil \crcr
9185       \c_math_toggle_token
9186       \underbrace
9187       {
9188         \hbox_to_wd:nn
9189         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9190         { }
9191       }
9192       \c_math_toggle_token
9193       \cr
9194       \bool_if:NTF \l_@@_tabular_bool
9195       { \begin { tabular } { c } #1 \end { tabular } }
9196       { $ \begin { array } { c } #1 \end { array } $ }
9197       \cr
9198     }
9199   }
9200   \group_end:
9201 }
9202 { }
9203 { }
9204 }

```

35 The commands HBrace et VBrace

```

9205 \hook_gput_code:nnn { begindocument } { . }
9206 {
9207   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9208   {
9209     \tikzset
9210     {
9211       nicematrix / brace / .style =
9212       {
9213         decoration = { brace , raise = -0.15 em } ,
9214         decorate ,
9215       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9216       nicematrix / mirrored-brace / .style =
9217       {
9218         nicematrix / brace ,
9219         decoration = mirror ,
9220       }
9221     }
9222   }

```

```
9223 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```
9224 \keys_define:nn { nicematrix / Hbrace }
9225 {
9226   color .code:n = ,
9227   horizontal-labels .code:n = ,
9228   shorten .code:n = ,
9229   shorten-start .code:n = ,
9230   shorten-end .code:n = ,
9231   unknown .code:n = \@@_error:n { Unknown-key-for-Hbrace }
9232 }
```

Here we need an “fully expandable” command.

```
9233 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9234 {
9235   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9236     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9237     { \@@_error:nn { Hbrace-not-allowed } { \Hbrace } }
9238 }
```

The following command must *not* be protected.

```
9239 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9240 {
9241   \int_compare:nNnTF { \c@iRow } < { \c_one_int }
9242   {
```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```
9243   \str_if_eq:nnTF { #2 } { * }
9244   {
9245     \NiceMatrixOptions { nullify-dots }
9246     \Ldots
9247     [
9248       line-style = nicematrix / brace ,
9249       #1 ,
9250       up =
9251       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9252     ]
9253   }
9254   {
9255     \Hdotsfor
9256     [
9257       line-style = nicematrix / brace ,
9258       #1 ,
9259       up =
9260       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9261     ]
9262     { #2 }
9263   }
9264 }
9265 {
9266   \str_if_eq:nnTF { #2 } { * }
9267   {
9268     \NiceMatrixOptions { nullify-dots }
9269     \Ldots
9270     [
9271       line-style = nicematrix / mirrored-brace ,
9272       #1 ,
9273       down =
9274       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9275     ]
9276   }
```

```

9277     {
9278       \Hdotsfor
9279       [
9280         line-style = nicematrix / mirrored-brace ,
9281         #1 ,
9282         down =
9283         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9284       ]
9285       { #2 }
9286     }
9287   }
9288   \keys_set:nn { nicematrix / Hbrace } { #1 }
9289 }

```

Here we need an “fully expandable” command.

```

9290 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9291 {
9292   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9293   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9294   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9295 }

```

The following command must *not* be protected.

```

9296 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9297 {
9298   \int_if_zero:nTF { \c@jCol }
9299   {
9300     \str_if_eq:nnTF { #2 } { * }
9301     {
9302       \NiceMatrixOptions { nullify-dots }
9303       \Vdots
9304       [
9305         line-style = nicematrix / mirrored-brace ,
9306         #1 ,
9307         down =
9308         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9309       ]
9310     }
9311     {
9312       \Vdotsfor
9313       [
9314         line-style = nicematrix / mirrored-brace ,
9315         #1 ,
9316         down =
9317         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9318       ]
9319       { #2 }
9320     }
9321   }
9322   {
9323     \str_if_eq:nnTF { #2 } { * }
9324     {
9325       \NiceMatrixOptions { nullify-dots }
9326       \Vdots
9327       [
9328         line-style = nicematrix / brace ,
9329         #1 ,
9330         up =
9331         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9332       ]
9333     }
9334     {
9335       \Vdotsfor
9336       [

```

```

9337         line-style = nicematrix / brace ,
9338         #1 ,
9339         up =
9340         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9341     ]
9342     { #2 }
9343 }
9344 }
9345 \keys_set:nn { nicematrix / Hbrace } { #1 }
9346 }

```

36 The command TikzEveryCell

```

9347 \bool_new:N \l_@@_not_empty_bool
9348 \bool_new:N \l_@@_empty_bool
9349
9350 \keys_define:nn { nicematrix / TikzEveryCell }
9351 {
9352     not-empty .code:n =
9353     \bool_lazy_or:nnTF
9354     { \l_@@_in_code_after_bool }
9355     { \g_@@_recreate_cell_nodes_bool }
9356     { \bool_set_true:N \l_@@_not_empty_bool }
9357     { \@@_error:n { detection-of-empty-cells } } ,
9358     not-empty .value_forbidden:n = true ,
9359     empty .code:n =
9360     \bool_lazy_or:nnTF
9361     { \l_@@_in_code_after_bool }
9362     { \g_@@_recreate_cell_nodes_bool }
9363     { \bool_set_true:N \l_@@_empty_bool }
9364     { \@@_error:n { detection-of-empty-cells } } ,
9365     empty .value_forbidden:n = true ,
9366     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9367 }
9368
9369
9370 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9371 {
9372     \IfPackageLoadedTF { tikz }
9373     {
9374         \group_begin:
9375         \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.

```

9376         \tl_set:Nn \l_tmpa_tl { { #2 } }
9377         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9378         { \@@_for_a_block:nnnnn ##1 }
9379         \@@_all_the_cells:
9380         \group_end:
9381     }
9382     { \@@_error:n { TikzEveryCell~without~tikz } }
9383 }
9384
9385 \tl_new:N \l_@@_i_tl
9386 \tl_new:N \l_@@_j_tl
9387
9388
9389 \cs_new_protected:Nn \@@_all_the_cells:
9390 {
9391     \int_step_inline:nn \c@iRow
9392     {

```

```

9393 \int_step_inline:nn \c@jCol
9394 {
9395   \cs_if_exist:cF { cell - ##1 - #####1 }
9396   {
9397     \clist_if_in:NcF \l_@@_corners_cells_clist
9398     { ##1 - #####1 }
9399     {
9400       \bool_set_false:N \l_tmpa_bool
9401       \cs_if_exist:cTF
9402       { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9403       {
9404         \bool_if:NF \l_@@_empty_bool
9405         { \bool_set_true:N \l_tmpa_bool }
9406       }
9407       {
9408         \bool_if:NF \l_@@_not_empty_bool
9409         { \bool_set_true:N \l_tmpa_bool }
9410       }
9411       \bool_if:NT \l_tmpa_bool
9412       {
9413         \@@_block_tikz:nnnnn
9414         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9415       }
9416     }
9417   }
9418 }
9419 }
9420 }
9421
9422 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9423 {
9424   \bool_if:NF \l_@@_empty_bool
9425   {
9426     \@@_block_tikz:nnnnn
9427     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9428   }
9429   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9430 }
9431
9432 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9433 {
9434   \int_step_inline:nnn { #1 } { #3 }
9435   {
9436     \int_step_inline:nnn { #2 } { #4 }
9437     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9438   }
9439 }

```

37 The command \ShowCellNames

```

9440 \NewDocumentCommand \@@_ShowCellNames { }
9441 {
9442   \bool_if:NT \l_@@_in_code_after_bool
9443   {
9444     \pgfpicture
9445     \pgfrememberpicturepositiononpagetrue
9446     \pgf@relevantforpicturesizefalse
9447     \pgfpathrectanglecorners
9448     { \@@_qpoint:n { 1 } }
9449     {
9450       \@@_qpoint:n
9451       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }

```

```

9452     }
9453     \pgfsetfillopacity { 0.75 }
9454     \pgfsetfillcolor { white }
9455     \pgfusepathqfill
9456     \endpgfpicture
9457 }
9458 \dim_gzero_new:N \g_@@_tmpc_dim
9459 \dim_gzero_new:N \g_@@_tmpd_dim
9460 \dim_gzero_new:N \g_@@_tmpe_dim
9461 \int_step_inline:nn { \c@iRow }
9462 {
9463     \bool_if:NTF \l_@@_in_code_after_bool
9464     {
9465         \pgfpicture
9466         \pgfrememberpicturepositiononpagetrue
9467         \pgf@relevantforpicturesizefalse
9468     }
9469     { \begin { pgfpicture } }
9470     \@@_qpoint:n { row - ##1 }
9471     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9472     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9473     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9474     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9475     \bool_if:NTF \l_@@_in_code_after_bool
9476     { \endpgfpicture }
9477     { \end { pgfpicture } }
9478     \int_step_inline:nn { \c@jCol }
9479     {
9480         \hbox_set:Nn \l_tmpa_box
9481         {
9482             \normalfont \Large \sffamily \bfseries
9483             \bool_if:NTF \l_@@_in_code_after_bool
9484             { \color { red } }
9485             { \color { red ! 50 } }
9486             ##1 - #####1
9487         }
9488         \bool_if:NTF \l_@@_in_code_after_bool
9489         {
9490             \pgfpicture
9491             \pgfrememberpicturepositiononpagetrue
9492             \pgf@relevantforpicturesizefalse
9493         }
9494         { \begin { pgfpicture } }
9495         \@@_qpoint:n { col - #####1 }
9496         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9497         \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9498         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9499         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9500         \bool_if:NTF \l_@@_in_code_after_bool
9501         { \endpgfpicture }
9502         { \end { pgfpicture } }
9503         \fp_set:Nn \l_tmpa_fp
9504         {
9505             \fp_min:nn
9506             {
9507                 \fp_min:nn
9508                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9509                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9510             }
9511             { 1.0 }
9512         }
9513         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9514         \pgfpicture

```



```

9515     \pgfrememberpicturepositiononpagetrue
9516     \pgf@relevantforpicturesizefalse
9517     \pgftransformshift
9518     {
9519         \pgfpoint
9520         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9521         { \dim_use:N \g_tmpa_dim }
9522     }
9523     \pgfnode
9524     { rectangle }
9525     { center }
9526     { \box_use:N \l_tmpa_box }
9527     { }
9528     { }
9529     \endpgfpicture
9530 }
9531 }
9532 }

```

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.

```

9533 \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.

```

9534 \bool_new:N \g_@@_footnote_bool

9535 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9536 {
9537     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9538     but-that-key-is-unknown. \\
9539     It-will-be-ignored. \\
9540     For-a-list-of-the-available-keys,~type-H~<return>.
9541 }
9542 {
9543     The-available-keys-are~(in~alphabetic~order):~
9544     footnote,~
9545     footnotehyper,~
9546     messages-for-Overleaf,~
9547     renew-dots~and~
9548     renew-matrix.
9549 }

9550 \keys_define:nn { nicematrix }
9551 {
9552     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9553     renew-dots .value_forbidden:n = true ,
9554     renew-matrix .code:n = \@@_renew_matrix: ,
9555     renew-matrix .value_forbidden:n = true ,
9556     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9557     footnote .bool_set:N = \g_@@_footnote_bool ,
9558     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9559     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9560 }
9561 \ProcessKeyOptions

```

```

9562 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9563 {
9564   You~can't~use~the~option~'footnote'~because~the~package~
9565   footnotehyper~has~already~been~loaded.~
9566   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9567   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9568   of~the~package~footnotehyper.\\
9569   The~package~footnote~won't~be~loaded.
9570 }
9571 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9572 {
9573   You~can't~use~the~option~'footnotehyper'~because~the~package~
9574   footnote~has~already~been~loaded.~
9575   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9576   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9577   of~the~package~footnote.\\
9578   The~package~footnotehyper~won't~be~loaded.
9579 }
9580 \bool_if:NT \g_@@_footnote_bool
9581 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9582   \IfClassLoadedTF { beamer }
9583   { \bool_set_false:N \g_@@_footnote_bool }
9584   {
9585     \IfPackageLoadedTF { footnotehyper }
9586     { \@@_error:n { footnote-with-footnotehyper-package } }
9587     { \usepackage { footnote } }
9588   }
9589 }
9590 \bool_if:NT \g_@@_footnotehyper_bool
9591 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9592   \IfClassLoadedTF { beamer }
9593   { \bool_set_false:N \g_@@_footnote_bool }
9594   {
9595     \IfPackageLoadedTF { footnote }
9596     { \@@_error:n { footnotehyper-with-footnote-package } }
9597     { \usepackage { footnotehyper } }
9598   }
9599   \bool_set_true:N \g_@@_footnote_bool
9600 }

```

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.

```

9601 \bool_new:N \l_@@_underscore_loaded_bool
9602 \IfPackageLoadedT { underscore }
9603 { \bool_set_true:N \l_@@_underscore_loaded_bool }

```

```

9604 \hook_gput_code:nnn { begindocument } { . }
9605 {
9606   \bool_if:NF \l_@@_underscore_loaded_bool
9607   {
9608     \IfPackageLoadedT { underscore }
9609     { \@@_error:n { underscore-after-nicematrix } }
9610   }
9611 }

```

40 Error messages of the package

```

9612 \str_const:Ne \c_@@_available_keys_str
9613 {
9614   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9615   { For~a~list~of~the~available~keys,~type~H~<return>. }
9616   { }
9617 }
9618 \seq_new:N \g_@@_types_of_matrix_seq
9619 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9620 {
9621   NiceMatrix ,
9622   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9623 }
9624 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9625 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_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`.

```

9626 \cs_new_protected:Npn \@@_error_too_much_cols:
9627 {
9628   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9629   { \@@_fatal:nn { too-much-cols-for-array } }
9630   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9631   { \@@_fatal:n { too-much-cols-for-matrix } }
9632   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9633   { \@@_fatal:n { too-much-cols-for-matrix } }
9634   \bool_if:NF \l_@@_last_col_without_value_bool
9635   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9636 }

```

The following command must *not* be protected since it's used in an error message.

```

9637 \cs_new:Npn \@@_message_hdotsfor:
9638 {
9639   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9640   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ is~incorrect. }
9641 }
9642 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9643 {
9644   Incompatible~options.\\
9645   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9646   The~output~will~not~be~reliable.
9647 }
9648 \@@_msg_new:nn { key~color~inside }
9649 {
9650   Key~deprecated.\\
9651   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~

```

```

9652     and-have-been-deprecated.\\
9653     You-won't-have-similar-message-till-the-end-of-the-document.
9654 }
9655 \@@_msg_new:nn { negative-weight }
9656 {
9657     Negative-weight.\\
9658     The-weight-of-the-'X'-columns-must-be-positive-and-you-have-used~
9659     the-value~' \int_use:N \l_@@_weight_int '.\\
9660     The-absolute-value-will-be-used.
9661 }
9662 \@@_msg_new:nn { last-col-not-used }
9663 {
9664     Column-not-used.\\
9665     The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
9666     in-your~\@@_full_name_env: .~
9667     However,~you~can~go-on.
9668 }
9669 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9670 {
9671     Too-much-columns.\\
9672     In-the-row~ \int_eval:n { \c@iRow },~
9673     you-try-to-use-more-columns~
9674     than-allowed-by-your~ \@@_full_name_env: .
9675     \@@_message_hdotsfor: \
9676     The-maximal-number-of-columns-is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9677     (plus-the-exterior-columns).~This-error-is-fatal.
9678 }
9679 \@@_msg_new:nn { too-much-cols-for-matrix }
9680 {
9681     Too-much-columns.\\
9682     In-the-row~ \int_eval:n { \c@iRow },~
9683     you-try-to-use-more-columns-than-allowed-by-your~ \@@_full_name_env: .
9684     \@@_message_hdotsfor: \
9685     Recall-that-the-maximal-number-of-columns-for-a-matrix~
9686     (excepted-the-potential-exterior-columns)-is-fixed-by-the~
9687     LaTeX-counter~'MaxMatrixCols'.~
9688     Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
9689     (use~ \token_to_str:N \setcounter \ to-change-that-value).~
9690     This-error-is-fatal.
9691 }
9692 \@@_msg_new:nn { too-much-cols-for-array }
9693 {
9694     Too-much-columns.\\
9695     In-the-row~ \int_eval:n { \c@iRow },~
9696     ~you-try-to-use-more-columns-than-allowed-by-your~
9697     \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
9698     \int_use:N \g_@@_static_num_of_col_int \
9699     \bool_if:nT
9700     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9701     { ~(plus-the-exterior-ones) }
9702     since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9703     This-error-is-fatal.
9704 }
9705 \@@_msg_new:nn { columns-not-used }
9706 {
9707     Columns-not-used.\\
9708     The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
9709     It-announces~ \int_use:N \g_@@_static_num_of_col_int \
9710     columns-but-you-only-used~ \int_use:N \c@jCol .\\
9711     The-columns-you-did-not-used-won't-be-created.\\
9712     You-won't-have-similar-warning-till-the-end-of-the-document.

```

```

9713 }
9714 \@@_msg_new:nn { empty-preamble }
9715 {
9716   Empty~preamble.\\
9717   The~preamble-of~your~ \@@_full_name_env: \ is~empty.\\
9718   This~error~is~fatal.
9719 }
9720 \@@_msg_new:nn { in~first~col }
9721 {
9722   Erroneous~use.\\
9723   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9724   That~command~will~be~ignored.
9725 }
9726 \@@_msg_new:nn { in~last~col }
9727 {
9728   Erroneous~use.\\
9729   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9730   That~command~will~be~ignored.
9731 }
9732 \@@_msg_new:nn { in~first~row }
9733 {
9734   Erroneous~use.\\
9735   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9736   That~command~will~be~ignored.
9737 }
9738 \@@_msg_new:nn { in~last~row }
9739 {
9740   Erroneous~use.\\
9741   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9742   That~command~will~be~ignored.
9743 }
9744 \@@_msg_new:nn { TopRule~without~booktabs }
9745 {
9746   Erroneous~use.\\
9747   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9748   That~command~will~be~ignored.
9749 }
9750 \@@_msg_new:nn { TopRule~without~tikz }
9751 {
9752   Erroneous~use.\\
9753   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9754   That~command~will~be~ignored.
9755 }
9756 \@@_msg_new:nn { caption~outside~float }
9757 {
9758   Key~caption~forbidden.\\
9759   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9760   environment~(such~as~\{table\}).~This~key~will~be~ignored.
9761 }
9762 \@@_msg_new:nn { short~caption~without~caption }
9763 {
9764   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9765   However,~your~'short~caption'~will~be~used~as~'caption'.
9766 }
9767 \@@_msg_new:nn { double~closing~delimiter }
9768 {
9769   Double~delimiter.\\
9770   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9771   delimiter.~This~delimiter~will~be~ignored.

```

```

9772 }
9773 \@@_msg_new:nn { delimiter~after~opening }
9774 {
9775   Double~delimiter.\\
9776   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9777   delimiter.~That~delimiter~will~be~ignored.
9778 }
9779 \@@_msg_new:nn { bad~option~for~line~style }
9780 {
9781   Bad~line~style.\\
9782   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9783   is~'standard'.~That~key~will~be~ignored.
9784 }
9785 \@@_msg_new:nn { corners~with~no~cell~nodes }
9786 {
9787   Incompatible~keys.\\
9788   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9789   is~in~force.\\
9790   If~you~go~on,~that~key~will~be~ignored.
9791 }
9792 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9793 {
9794   Incompatible~keys.\\
9795   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9796   is~in~force.\\
9797   If~you~go~on,~those~extra~nodes~won't~be~created.
9798 }
9799 \@@_msg_new:nn { Identical~notes~in~caption }
9800 {
9801   Identical~tabular~notes.\\
9802   You~can't~put~several~notes~with~the~same~content~in~
9803   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9804   If~you~go~on,~the~output~will~probably~be~erroneous.
9805 }
9806 \@@_msg_new:nn { tabularnote~below~the~tabular }
9807 {
9808   \token_to_str:N \tabularnote \ forbidden\\
9809   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9810   of~your~tabular~because~the~caption~will~be~composed~below~
9811   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9812   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9813   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9814   no~similar~error~will~raised~in~this~document.
9815 }
9816 \@@_msg_new:nn { Unknown~key~for~rules }
9817 {
9818   Unknown~key.\\
9819   There~is~only~two~keys~available~here:~width~and~color.\\
9820   Your~key~' \l_keys_key_str ' ~will~be~ignored.
9821 }
9822 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9823 {
9824   Unknown~key.\\
9825   You~have~used~the~key~' \l_keys_key_str ' ~but~the~only~
9826   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9827   and~ \token_to_str:N \Vbrace \ are:~'color',~
9828   'horizontal~labels',~'shorten'~'shorten~end'~
9829   and~'shorten~start'.
9830 }
9831 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }

```

```

9832 {
9833   Unknown~key.\\
9834   There~is~only~two~keys~available~here::~
9835   'empty'~and~'not-empty'.\\
9836   Your~key~' \l_keys_key_str ' ~will~be~ignored.
9837 }
9838 \@@_msg_new:nn { Unknown~key~for~rotate }
9839 {
9840   Unknown~key.\\
9841   The~only~key~available~here~is~'c'.\\
9842   Your~key~' \l_keys_key_str ' ~will~be~ignored.
9843 }
9844 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9845 {
9846   Unknown~key.\\
9847   The~key~' \l_keys_key_str ' ~is~unknown~in~a~'custom~line'.~
9848   It~you~go~on,~you~will~probably~have~other~errors. \\
9849   \c_@@_available_keys_str
9850 }
9851 {
9852   The~available~keys~are~(in~alphabetic~order):~
9853   ccommand,~
9854   color,~
9855   command,~
9856   dotted,~
9857   letter,~
9858   multiplicity,~
9859   sep-color,~
9860   tikz,~and~total~width.
9861 }
9862 \@@_msg_new:nnn { Unknown~key~for~xdots }
9863 {
9864   Unknown~key.\\
9865   The~key~' \l_keys_key_str ' ~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9866   \c_@@_available_keys_str
9867 }
9868 {
9869   The~available~keys~are~(in~alphabetic~order):~
9870   'color',~
9871   'horizontal-labels',~
9872   'inter',~
9873   'line-style',~
9874   'radius',~
9875   'shorten',~
9876   'shorten-end'~and~'shorten-start'.
9877 }
9878 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9879 {
9880   Unknown~key.\\
9881   As~for~now,~there~is~only~two~keys~available~here::~'cols'~and~'respect-blocks'~
9882   (and~you~try~to~use~' \l_keys_key_str ' )\\
9883   That~key~will~be~ignored.
9884 }
9885 \@@_msg_new:nn { label~without~caption }
9886 {
9887   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9888   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9889 }
9890 \@@_msg_new:nn { W~warning }
9891 {
9892   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~

```

```

9893     (row~ \int_use:N \c@iRow ).
9894 }
9895 \@@_msg_new:nn { Construct-too-large }
9896 {
9897     Construct-too-large.\
9898     Your-command~ \token_to_str:N #1
9899     can't-be-drawn-because-your-matrix-is-too-small.\
9900     That-command-will-be-ignored.
9901 }
9902 \@@_msg_new:nn { underscore-after-nicematrix }
9903 {
9904     Problem-with-'underscore'.\
9905     The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9906     You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
9907     ' \token_to_str:N \Cdots \token_to_str:N _
9908     \{ n \token_to_str:N \text \{ ~times \} \}' .
9909 }
9910 \@@_msg_new:nn { ampersand-in-light-syntax }
9911 {
9912     Ampersand-forbidden.\
9913     You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns-because~
9914     ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9915 }
9916 \@@_msg_new:nn { double-backslash-in-light-syntax }
9917 {
9918     Double-backslash-forbidden.\
9919     You-can't-use~ \token_to_str:N \
9920     ~to-separate-rows-because-the-key~'light-syntax'~
9921     is-in-force.~You-must-use-the-character~' \l_@@_end_of_row_tl '~
9922     (set-by-the-key~'end-of-row').~This-error-is-fatal.
9923 }
9924 \@@_msg_new:nn { hlines-with-color }
9925 {
9926     Incompatible-keys.\
9927     You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9928     \token_to_str:N \Block \ when-the-key~'color'~or~'draw'~is-used.\
9929     However,~you-can-put-several-commands~ \token_to_str:N \Block.\
9930     Your-key-will-be-discarded.
9931 }
9932 \@@_msg_new:nn { bad-value-for-baseline }
9933 {
9934     Bad-value-for-baseline.\
9935     The-value-given-to~'baseline'~( \int_use:N \l_tmpa_int )~is-not~
9936     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int~and~
9937     \int_use:N \g_@@_row_total_int \ or-equal-to~'t',~'c'~or~'b'~or~of~
9938     the-form~'line-i'.\
9939     A~value-of~1~will-be-used.
9940 }
9941 \@@_msg_new:nn { detection-of-empty-cells }
9942 {
9943     Problem-with~'not-empty'~\
9944     For-technical-reasons,~you-must-activate~
9945     'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
9946     in-order-to-use-the-key~' \l_keys_key_str '~.\
9947     That-key-will-be-ignored.
9948 }
9949 \@@_msg_new:nn { siunitx-not-loaded }
9950 {
9951     siunitx-not-loaded\
9952     You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\

```



```

9953     That~error~is~fatal.
9954 }

9955 \@@_msg_new:nn { Invalid~name }
9956 {
9957     Invalid~name.\\
9958     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9959     \SubMatrix \ of~your~ \@@_full_name_env: .\\
9960     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9961     This~key~will~be~ignored.
9962 }

9963 \@@_msg_new:nn { Hbrace~not~allowed }
9964 {
9965     Command~not~allowed.\\
9966     You~can't~use~the~command~ \token_to_str:N #1
9967     because~you~have~not~loaded~
9968     \IfPackageLoadedTF { tikz }
9969     { the~TikZ~library~'decorations.pathreplacing'~.~Use~ }
9970     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9971     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9972     That~command~will~be~ignored.
9973 }

9974 \@@_msg_new:nn { Vbrace~not~allowed }
9975 {
9976     Command~not~allowed.\\
9977     You~can't~use~the~command~ \token_to_str:N \Vbrace \
9978     because~you~have~not~loaded~TikZ~
9979     and~the~TikZ~library~'decorations.pathreplacing'~.\\
9980     Use: ~\token_to_str:N \usepackage \{tikz\}~
9981     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9982     That~command~will~be~ignored.
9983 }

9984 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9985 {
9986     Wrong~line.\\
9987     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9988     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
9989     number~is~not~valid.~It~will~be~ignored.
9990 }

9991 \@@_msg_new:nn { Impossible~delimiter }
9992 {
9993     Impossible~delimiter.\\
9994     It's~impossible~to~draw~the~#1~delimiter~of~your~
9995     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
9996     in~that~column.
9997     \bool_if:NT \l_@@_submatrix_slim_bool
9998     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9999     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10000 }

10001 \@@_msg_new:nnn { width~without~X~columns }
10002 {
10003     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10004     the~preamble~(' \g_@@_user_preamble_tl '~of~your~ \@@_full_name_env: .\\
10005     That~key~will~be~ignored.
10006 }
10007 {
10008     This~message~is~the~message~'width~without~X~columns'~
10009     of~the~module~'nicematrix'.~
10010     The~experimented~users~can~disable~that~message~with~
10011     \token_to_str:N \msg_redirect_name:nnn .\\
10012 }
10013

```

```

10014 \@@_msg_new:nn { key-multiplicity-with-dotted }
10015 {
10016   Incompatible-keys. \\
10017   You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
10018   in~a~'custom-line'.~They~are~incompatible. \\
10019   The-key~'multiplicity'~will-be-discarded.
10020 }
10021 \@@_msg_new:nn { empty-environment }
10022 {
10023   Empty~environment.\\
10024   Your~ \@@_full_name_env: \ is-empty.~This-error-is-fatal.
10025 }
10026 \@@_msg_new:nn { No-letter-and-no-command }
10027 {
10028   Erroneous-use.\\
10029   Your-use-of~'custom-line'~is-no-op~since~you~don't~have~used~the~
10030   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10031   ~'ccommand'~(to~draw~horizontal~rules).\\
10032   However,~you~can~go~on.
10033 }
10034 \@@_msg_new:nn { Forbidden-letter }
10035 {
10036   Forbidden-letter.\\
10037   You-can't-use-the-letter~'#1'~for~a~customized-line.~
10038   It~will~be~ignored.\\
10039   The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10040 }
10041 \@@_msg_new:nn { Several-letters }
10042 {
10043   Wrong-name.\\
10044   You-must-use-only-one-letter~as~value~for~the~key~'letter'~(and~you~
10045   have~used~' \l_@@_letter_str ').\\
10046   It~will~be~ignored.
10047 }
10048 \@@_msg_new:nn { Delimiter-with-small }
10049 {
10050   Delimiter~forbidden.\\
10051   You-can't-put~a~delimiter~in~the~preamble~of~your~
10052   \@@_full_name_env: \
10053   because~the~key~'small'~is~in~force.\\
10054   This-error-is-fatal.
10055 }
10056 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
10057 {
10058   Unknown~cell.\\
10059   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10060   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10061   can't-be-executed~because~a~cell~doesn't~exist.\\
10062   This~command~ \token_to_str:N \line \ will~be~ignored.
10063 }
10064 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
10065 {
10066   Duplicate-name.\\
10067   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10068   in~this~ \@@_full_name_env: .\\
10069   This~key~will~be~ignored.\\
10070   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10071   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10072 }
10073 {
10074   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~

```

```

10075 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10076 }
10077 \@@_msg_new:nn { r~or~l~with~preamble }
10078 {
10079   Erroneous~use.\\
10080   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10081   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10082   your~ \@@_full_name_env: .\\
10083   This~key~will~be~ignored.
10084 }
10085 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10086 {
10087   Erroneous~use.\\
10088   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10089   the~array.~This~error~is~fatal.
10090 }
10091 \@@_msg_new:nn { bad~corner }
10092 {
10093   Bad~corner.\\
10094   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10095   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10096   This~specification~of~corner~will~be~ignored.
10097 }
10098 \@@_msg_new:nn { bad~border }
10099 {
10100   Bad~border.\\
10101   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10102   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10103   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10104   also~use~the~key~'tikz'
10105   \IfPackageLoadedF { tikz }
10106   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10107   This~specification~of~border~will~be~ignored.
10108 }
10109 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10110 {
10111   TikZ~not~loaded.\\
10112   You~can't~use~ \token_to_str:N \TikzEveryCell \
10113   because~you~have~not~loaded~tikz.~
10114   This~command~will~be~ignored.
10115 }
10116 \@@_msg_new:nn { tikz~key~without~tikz }
10117 {
10118   TikZ~not~loaded.\\
10119   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10120   \Block '~because~you~have~not~loaded~tikz.~
10121   This~key~will~be~ignored.
10122 }
10123 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10124 {
10125   Erroneous~use.\\
10126   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10127   'last~col'~without~value.\\
10128   However,~you~can~go~on~for~this~time~
10129   (the~value~' \l_keys_value_tl '~will~be~ignored).
10130 }
10131 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10132 {
10133   Erroneous~use. \\
10134   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~

```

```

10135     'last-col'~without~value. \\
10136     However,~you~can~go~on~for~this~time~
10137     (the~value~' \l_keys_value_tl '~will~be~ignored).
10138 }
10139 \@@_msg_new:nn { Block-too-large-1 }
10140 {
10141     Block-too-large. \\
10142     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10143     too~small~for~that~block. \\
10144     This~block~and~maybe~others~will~be~ignored.
10145 }
10146 \@@_msg_new:nn { Block-too-large-2 }
10147 {
10148     Block-too-large. \\
10149     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10150     \g_@@_static_num_of_col_int \
10151     columns~but~you~use~only~ \int_use:N \c_jCol \ and~that's~why~a~block~
10152     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10153     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10154     This~block~and~maybe~others~will~be~ignored.
10155 }
10156 \@@_msg_new:nn { unknown~column~type }
10157 {
10158     Bad~column~type. \\
10159     The~column~type~'#1'~in~your~ \@@_full_name_env: \
10160     is~unknown. \\
10161     This~error~is~fatal.
10162 }
10163 \@@_msg_new:nn { unknown~column~type-S }
10164 {
10165     Bad~column~type. \\
10166     The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10167     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10168     load~that~package. \\
10169     This~error~is~fatal.
10170 }
10171 \@@_msg_new:nn { tabularnote~forbidden }
10172 {
10173     Forbidden~command. \\
10174     You~can't~use~the~command~ \token_to_str:N \tabularnote \
10175     ~here.~This~command~is~available~only~in~
10176     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10177     the~argument~of~a~command~\token_to_str:N \caption \ included~
10178     in~an~environment~\{table\}. \\
10179     This~command~will~be~ignored.
10180 }
10181 \@@_msg_new:nn { borders~forbidden }
10182 {
10183     Forbidden~key.\\
10184     You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10185     because~the~option~'rounded~corners'~
10186     is~in~force~with~a~non-zero~value.\\
10187     This~key~will~be~ignored.
10188 }
10189 \@@_msg_new:nn { bottomrule~without~booktabs }
10190 {
10191     booktabs~not~loaded.\\
10192     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10193     loaded~'booktabs'.\\
10194     This~key~will~be~ignored.
10195 }

```

```

10196 \@@_msg_new:nn { enumitem~not~loaded }
10197 {
10198   enumitem~not~loaded. \\
10199   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10200   ~because~you~haven't~loaded~'enumitem'. \\
10201   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10202   ignored~in~the~document.
10203 }
10204 \@@_msg_new:nn { tikz~without~tikz }
10205 {
10206   Tikz~not~loaded. \\
10207   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10208   loaded.~If~you~go~on,~that~key~will~be~ignored.
10209 }
10210 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10211 {
10212   Tikz~not~loaded. \\
10213   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10214   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10215   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10216   use~that~custom~line.
10217 }
10218 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10219 {
10220   Tikz~not~loaded. \\
10221   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10222   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10223   That~key~will~be~ignored.
10224 }
10225 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10226 {
10227   Erroneous~use.\\
10228   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10229   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10230   The~key~'color'~will~be~discarded.
10231 }
10232 \@@_msg_new:nn { Wrong~last~row }
10233 {
10234   Wrong~number.\\
10235   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10236   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10237   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10238   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10239   without~value~(more~compilations~might~be~necessary).
10240 }
10241 \@@_msg_new:nn { Yet~in~env }
10242 {
10243   Nested~environments.\\
10244   Environments~of~nicematrix~can't~be~nested.\\
10245   This~error~is~fatal.
10246 }
10247 \@@_msg_new:nn { Outside~math~mode }
10248 {
10249   Outside~math~mode.\\
10250   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10251   (and~not~in~ \token_to_str:N \vcenter ).\\
10252   This~error~is~fatal.
10253 }
10254 \@@_msg_new:nn { One~letter~allowed }
10255 {

```

```

10256     Bad-name.\\
10257     The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10258     you-have-used~' \l_keys_value_tl '~.\\
10259     It-will-be-ignored.
10260 }
10261 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10262 {
10263     Environment~\{TabularNote\}~forbidden.\\
10264     You-must-use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10265     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10266     This-environment~\{TabularNote\}~will-be-ignored.
10267 }
10268 \@@_msg_new:nn { varwidth~not~loaded }
10269 {
10270     varwidth~not~loaded.\\
10271     You-can't-use-the-column-type~'V'~because~'varwidth'~is~not~
10272     loaded.\\
10273     Your~column~will~behave~like~'p'.
10274 }
10275 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10276 {
10277     Unknown~key.\\
10278     Your~key~' \l_keys_key_str '~is-unknown~for~a~rule.\\
10279     \c_@@_available_keys_str
10280 }
10281 {
10282     The-available-keys-are~(in~alphabetic~order):~
10283     color,~
10284     dotted,~
10285     multiplicity,~
10286     sep-color,~
10287     tikz,~and~total-width.
10288 }
10289
10290 \@@_msg_new:nnn { Unknown~key~for~Block }
10291 {
10292     Unknown~key. \\
10293     The~key~' \l_keys_key_str '~is-unknown~for~the~command~
10294     \token_to_str:N \Block . \\
10295     It-will-be-ignored. \\
10296     \c_@@_available_keys_str
10297 }
10298 {
10299     The-available-keys-are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10300     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10301     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10302     and~vlines.
10303 }
10304 \@@_msg_new:nnn { Unknown~key~for~Brace }
10305 {
10306     Unknown~key.\\
10307     The~key~' \l_keys_key_str '~is-unknown~for~the~commands~
10308     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10309     It-will-be-ignored. \\
10310     \c_@@_available_keys_str
10311 }
10312 {
10313     The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
10314     right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10315     right-shorten)~and~yshift.
10316 }

```

```

10317 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10318 {
10319   Unknown~key.\\
10320   The~key~' \l_keys_key_str '~is~unknown.\\
10321   It~will~be~ignored. \\
10322   \c_@@_available_keys_str
10323 }
10324 {
10325   The~available~keys~are~(in~alphabetic~order):~
10326   delimiters/color,~
10327   rules~(with~the~subkeys~'color'~and~'width'),~
10328   sub-matrix~(several~subkeys)~
10329   and~xdots~(several~subkeys).~
10330   The~latter~is~for~the~command~ \token_to_str:N \line .
10331 }
10332 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10333 {
10334   Unknown~key.\\
10335   The~key~' \l_keys_key_str '~is~unknown.\\
10336   It~will~be~ignored. \\
10337   \c_@@_available_keys_str
10338 }
10339 {
10340   The~available~keys~are~(in~alphabetic~order):~
10341   create-cell-nodes,~
10342   delimiters/color~and~
10343   sub-matrix~(several~subkeys).
10344 }
10345 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10346 {
10347   Unknown~key.\\
10348   The~key~' \l_keys_key_str '~is~unknown.\\
10349   That~key~will~be~ignored. \\
10350   \c_@@_available_keys_str
10351 }
10352 {
10353   The~available~keys~are~(in~alphabetic~order):~
10354   'delimiters/color',~
10355   'extra-height',~
10356   'hlines',~
10357   'hvlines',~
10358   'left-xshift',~
10359   'name',~
10360   'right-xshift',~
10361   'rules'~(with~the~subkeys~'color'~and~'width'),~
10362   'slim',~
10363   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10364   and~'right-xshift').\\
10365 }
10366 \@@_msg_new:nnn { Unknown~key~for~notes }
10367 {
10368   Unknown~key.\\
10369   The~key~' \l_keys_key_str '~is~unknown.\\
10370   That~key~will~be~ignored. \\
10371   \c_@@_available_keys_str
10372 }
10373 {
10374   The~available~keys~are~(in~alphabetic~order):~
10375   bottomrule,~
10376   code-after,~
10377   code-before,~
10378   detect-duplicates,~
10379   enumitem-keys,~

```

```

10380     enumitem-keys-para,~
10381     para,~
10382     label-in-list,~
10383     label-in-tabular~and~
10384     style.
10385 }
10386 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10387 {
10388     Unknown~key.\\
10389     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10390     \token_to_str:N \RowStyle . \\
10391     That~key~will~be~ignored. \\
10392     \c_@@_available_keys_str
10393 }
10394 {
10395     The~available~keys~are~(in~alphabetic~order):~
10396     bold,~
10397     cell-space-top-limit,~
10398     cell-space-bottom-limit,~
10399     cell-space-limits,~
10400     color,~
10401     fill~(alias:~rowcolor),~
10402     nb-rows,~
10403     opacity~and~
10404     rounded-corners.
10405 }
10406 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10407 {
10408     Unknown~key.\\
10409     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10410     \token_to_str:N \NiceMatrixOptions . \\
10411     That~key~will~be~ignored. \\
10412     \c_@@_available_keys_str
10413 }
10414 {
10415     The~available~keys~are~(in~alphabetic~order):~
10416     &-in-blocks,~
10417     allow-duplicate-names,~
10418     ampersand-in-blocks,~
10419     caption-above,~
10420     cell-space-bottom-limit,~
10421     cell-space-limits,~
10422     cell-space-top-limit,~
10423     code-for-first-col,~
10424     code-for-first-row,~
10425     code-for-last-col,~
10426     code-for-last-row,~
10427     corners,~
10428     custom-key,~
10429     create-extra-nodes,~
10430     create-medium-nodes,~
10431     create-large-nodes,~
10432     custom-line,~
10433     delimiters~(several~subkeys),~
10434     end-of-row,~
10435     first-col,~
10436     first-row,~
10437     hlines,~
10438     hvlines,~
10439     hvlines-except-borders,~
10440     last-col,~
10441     last-row,~
10442     left-margin,~

```



```

10443 light-syntax,~
10444 light-syntax-expanded,~
10445 matrix/columns-type,~
10446 no-cell-nodes,~
10447 notes~(several~subkeys),~
10448 nullify-dots,~
10449 pgf-node-code,~
10450 renew-dots,~
10451 renew-matrix,~
10452 respect-arraystretch,~
10453 rounded-corners,~
10454 right-margin,~
10455 rules~(with~the~subkeys~'color'~and~'width'),~
10456 small,~
10457 sub-matrix~(several~subkeys),~
10458 vlines,~
10459 xdots~(several~subkeys).
10460 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10461 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10462 {
10463   Unknown~key.\\
10464   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10465   \{NiceArray\}. \\
10466   That~key~will~be~ignored. \\
10467   \c_@@_available_keys_str
10468 }
10469 {
10470   The~available~keys~are~(in~alphabetic~order):~
10471   &~in~blocks,~
10472   ampersand~in~blocks,~
10473   b,~
10474   baseline,~
10475   c,~
10476   cell-space-bottom-limit,~
10477   cell-space-limits,~
10478   cell-space-top-limit,~
10479   code~after,~
10480   code~for~first~col,~
10481   code~for~first~row,~
10482   code~for~last~col,~
10483   code~for~last~row,~
10484   columns~width,~
10485   corners,~
10486   create~extra~nodes,~
10487   create~medium~nodes,~
10488   create~large~nodes,~
10489   extra~left~margin,~
10490   extra~right~margin,~
10491   first~col,~
10492   first~row,~
10493   hlines,~
10494   hvlines,~
10495   hvlines~except~borders,~
10496   last~col,~
10497   last~row,~
10498   left~margin,~
10499   light-syntax,~
10500   light-syntax-expanded,~
10501   name,~
10502   no~cell~nodes,~
10503   nullify~dots,~

```

```

10504 pgf-node-code,~
10505 renew-dots,~
10506 respect-arraystretch,~
10507 right-margin,~
10508 rounded-corners,~
10509 rules~(with~the~subkeys~'color'~and~'width'),~
10510 small,~
10511 t,~
10512 vl原因,~
10513 xdots/color,~
10514 xdots/shorten-start,~
10515 xdots/shorten-end,~
10516 xdots/shorten-and~
10517 xdots/line-style.
10518 }

```

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`).

```

10519 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10520 {
10521   Unknown~key.\\
10522   The~key~' \l_keys_key_str 'is~unknown~for~the~
10523   \@@_full_name_env: . \\
10524   That~key~will~be~ignored. \\
10525   \c_@@_available_keys_str
10526 }
10527 {
10528   The~available~keys~are~(in~alphabetic~order):~
10529   &-in-blocks,~
10530   ampersand-in-blocks,~
10531   b,~
10532   baseline,~
10533   c,~
10534   cell-space-bottom-limit,~
10535   cell-space-limits,~
10536   cell-space-top-limit,~
10537   code-after,~
10538   code-for-first-col,~
10539   code-for-first-row,~
10540   code-for-last-col,~
10541   code-for-last-row,~
10542   columns-type,~
10543   columns-width,~
10544   corners,~
10545   create-extra-nodes,~
10546   create-medium-nodes,~
10547   create-large-nodes,~
10548   extra-left-margin,~
10549   extra-right-margin,~
10550   first-col,~
10551   first-row,~
10552   hlines,~
10553   hvlines,~
10554   hvlines-except-borders,~
10555   l,~
10556   last-col,~
10557   last-row,~
10558   left-margin,~
10559   light-syntax,~
10560   light-syntax-expanded,~
10561   name,~
10562   no-cell-nodes,~
10563   nullify-dots,~
10564   pgf-node-code,~

```

```

10565     r,~
10566     renew-dots,~
10567     respect-arraystretch,~
10568     right-margin,~
10569     rounded-corners,~
10570     rules~(with~the~subkeys~'color'~and~'width'),~
10571     small,~
10572     t,~
10573     vlines,~
10574     xdots/color,~
10575     xdots/shorten-start,~
10576     xdots/shorten-end,~
10577     xdots/shorten-and~
10578     xdots/line-style.
10579 }
10580 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10581 {
10582     Unknown-key.\\
10583     The-key~' \l_keys_key_str '~is-unknown-for-the-environment~
10584     \{NiceTabular\}. \\
10585     That~key~will~be~ignored. \\
10586     \c_@@_available_keys_str
10587 }
10588 {
10589     The-available-keys-are~(in~alphabetic~order):~
10590     &-in-blocks,~
10591     ampersand-in-blocks,~
10592     b,~
10593     baseline,~
10594     c,~
10595     caption,~
10596     cell-space-bottom-limit,~
10597     cell-space-limits,~
10598     cell-space-top-limit,~
10599     code-after,~
10600     code-for-first-col,~
10601     code-for-first-row,~
10602     code-for-last-col,~
10603     code-for-last-row,~
10604     columns-width,~
10605     corners,~
10606     custom-line,~
10607     create-extra-nodes,~
10608     create-medium-nodes,~
10609     create-large-nodes,~
10610     extra-left-margin,~
10611     extra-right-margin,~
10612     first-col,~
10613     first-row,~
10614     hlines,~
10615     hvlines,~
10616     hvlines-except-borders,~
10617     label,~
10618     last-col,~
10619     last-row,~
10620     left-margin,~
10621     light-syntax,~
10622     light-syntax-expanded,~
10623     name,~
10624     no-cell-nodes,~
10625     notes~(several~subkeys),~
10626     nullify-dots,~
10627     pgf-node-code,~

```

```

10628     renew-dots,~
10629     respect-arraystretch,~
10630     right-margin,~
10631     rounded-corners,~
10632     rules~(with~the~subkeys~'color'~and~'width'),~
10633     short-caption,~
10634     t,~
10635     tabularnote,~
10636     vlines,~
10637     xdots/color,~
10638     xdots/shorten-start,~
10639     xdots/shorten-end,~
10640     xdots/shorten-and~
10641     xdots/line-style.
10642 }
10643 \@@_msg_new:nnn { Duplicate-name }
10644 {
10645   Duplicate-name.\\
10646   The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
10647   the-same-environment-name-twice.~You-can-go-on,~but,~
10648   maybe,~you-will-have-incorrect-results-especially~
10649   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10650   message-again,~use-the-key~'allow-duplicate-names'~in~
10651   ' \token_to_str:N \NiceMatrixOptions '.\\
10652   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10653     { For-a-list-of-the-names-already-used,~type-H<return>. }
10654 }
10655 {
10656   The-names~already-defined-in~this~document~are:~
10657   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10658 }
10659 \@@_msg_new:nn { Option~auto~for~columns-width }
10660 {
10661   Erroneous-use.\\
10662   You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10663   That-key-will-be-ignored.
10664 }
10665 \@@_msg_new:nn { NiceTabularX~without~X }
10666 {
10667   NiceTabularX~without~X.\\
10668   You-should-not-use~\{NiceTabularX\}~without~X~columns.\\
10669   However,~you-can-go-on.
10670 }
10671 \@@_msg_new:nn { Preamble~forgotten }
10672 {
10673   Preamble~forgotten.\\
10674   You-have-probably-forgotten-the-preamble-of-your~
10675   \@@_full_name_env: . \\
10676   This-error-is-fatal.
10677 }
10678 \@@_msg_new:nn { Invalid~col~number }
10679 {
10680   Invalid~column~number.\\
10681   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10682   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10683 }
10684 \@@_msg_new:nn { Invalid~row~number }
10685 {
10686   Invalid~row~number.\\
10687   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10688   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10689 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
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>	37
9	The <code>\CodeBefore</code>	51
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	75
13	The environment <code>{NiceMatrix}</code> and its variants	93
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
15	After the construction of the array	95
16	We draw the dotted lines	102
17	The actual instructions for drawing the dotted lines with Tikz	116
18	User commands available in the new environments	121
19	The command <code>\line</code> accessible in code-after	128
20	The command <code>\RowStyle</code>	129
21	Colors of cells, rows and columns	132
22	The vertical and horizontal rules	144
23	The empty corners	161
24	The environment <code>{NiceMatrixBlock}</code>	163
25	The extra nodes	164
26	The blocks	169
27	How to draw the dotted lines transparently	194
28	Automatic arrays	194
29	The redefinition of the command <code>\dotfill</code>	195
30	The command <code>\diagbox</code>	196

31	The keyword <code>\CodeAfter</code>	197
32	The delimiters in the preamble	198
33	The command <code>\SubMatrix</code>	199
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	208
35	The commands <code>HBrace</code> et <code>VBrace</code>	211
36	The command <code>TikzEveryCell</code>	214
37	The command <code>\ShowCellNames</code>	215
38	We process the options at package loading	217
39	About the package underscore	218
40	Error messages of the package	219