

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

April 28, 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 7.1c of `nicematrix`, at the date of 2025/04/28.

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:nmw { #1 } { #2 } }
94     { #1 { #2 } }
95 }
96
97 \cs_new_protected:Npn \@@_collect_options:nmw #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:nNnT { \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:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
220     { \cs_gset:Npn \CT@drs@ { \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:en { \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_CTrsc: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_CTrsc: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[l]{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 list `\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 raise then 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`). You will use that flag for the blocks.

```
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 contain 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 sum of the weights of all the X-columns in the preamble.

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

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

```
409 \bool_new:N \l_@@_X_columns_aux_bool
410 \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.

```
411 \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).

```
412 \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).

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

```
414 \tl_new:N \l_@@_code_before_tl
415 \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).

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

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

```
417 \dim_new:N \l_@@_x_initial_dim
418 \dim_new:N \l_@@_y_initial_dim
419 \dim_new:N \l_@@_x_final_dim
420 \dim_new:N \l_@@_y_final_dim

421 \dim_new:N \g_@@_dp_row_zero_dim
422 \dim_new:N \g_@@_ht_row_zero_dim
423 \dim_new:N \g_@@_ht_row_one_dim
424 \dim_new:N \g_@@_dp_ante_last_row_dim
425 \dim_new:N \g_@@_ht_last_row_dim
426 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
427 \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”.

```
428 \dim_new:N \g_@@_width_last_col_dim
429 \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.

```
430 \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!).

```
431 \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.

```
432 \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}`.

```
433 \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.

```
434 \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`).

```
435 \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`).

```
436 \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.

```
437 \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).

```
438 \seq_new:N \g_@@multicolumn_cells_seq
439 \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.

```
440 \int_new:N \g_@@ddots_int
441 \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.

```
442 \dim_new:N \g_@@delta_x_one_dim
443 \dim_new:N \g_@@delta_y_one_dim
444 \dim_new:N \g_@@delta_x_two_dim
445 \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).

```
446 \int_new:N \l_@@row_min_int
447 \int_new:N \l_@@row_max_int
448 \int_new:N \l_@@col_min_int
449 \int_new:N \l_@@col_max_int

450 \int_new:N \l_@@initial_i_int
451 \int_new:N \l_@@initial_j_int
452 \int_new:N \l_@@final_i_int
453 \int_new:N \l_@@final_j_int
```

The following counters will be used when drawing the rules.

```
454 \int_new:N \l_@@start_int
455 \int_set_eq:NN \l_@@start_int \c_one_int
456 \int_new:N \l_@@end_int
457 \int_new:N \l_@@local_start_int
458 \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.

```
459 \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).

```
460 \int_new:N \g_@@static_num_of_col_int
```

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

```
461 \tl_new:N \l_@@fill_tl
462 \tl_new:N \l_@@opacity_tl
463 \tl_new:N \l_@@draw_tl
```

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


```

464 \seq_new:N \l_@@_tikz_seq
465 \clist_new:N \l_@@_borders_clist
466 \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.

```

467 \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`.

```

468 \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.

```

469 \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.

```

470 \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`).

```

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

```

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

```

475 \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).

```

476 \str_new:N \l_@@_vpos_block_str

```

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

```

477 \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).

```

478 \bool_new:N \l_@@_vlines_block_bool
479 \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.

```

480 \int_new:N \g_@@_block_box_int

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

```

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

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

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

```
489 \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`).

```
490 \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.

```
491 \int_new:N \l_@@_first_row_int
492 \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.

```
493 \int_new:N \l_@@_first_col_int
494 \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).

```
495 \int_new:N \l_@@_last_row_int
496 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
498 \bool_new:N \l_@@_last_col_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.

- **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 .

```
499 \int_new:N \l_@@_last_col_int
500 \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.

```
501 \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`).

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

Some utilities

```
503 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
504 {
```

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

```
505 \def \l_tmpa_tl { #1 }
506 \def \l_tmpb_tl { #2 }
507 }
```

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.

```
508 \cs_new_protected:Npn \@@_expand_clist:N #1
509 {
510 \clist_if_in:NnF #1 { all }
511 {
512 \clist_clear:N \l_tmpa_clist
513 \clist_map_inline:Nn #1
514 {
```

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

```
515 \tl_if_in:nnTF { ##1 } { - }
516 { \@@_cut_on_hyphen:w ##1 \q_stop }
517 {
```

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

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

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.

```

527 \hook_gput_code:nnn { begindocument } { . }
528   {
529     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
530     \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
531     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
532   }

```

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`).
- 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.

```

533 \newcounter { tabularnote }

```

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

```

534 \int_new:N \g_@@_tabularnote_int
535 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

536 \seq_new:N \g_@@_notes_seq
537 \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.

```

538 \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.

```

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

```

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

```

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

```

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

```

547 \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`.

```

548 \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`.

```

549 \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`).

```

550 \hook_gput_code:nnn { begindocument } { . }
551 {
552   \IfPackageLoadedTF { enumitem }
553   {

```

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.

```

554     \newlist { tabularnotes } { enumerate } { 1 }
555     \setlist [ tabularnotes ]
556     {
557       topsep = 0pt ,
558       noitemsep ,
559       leftmargin = * ,
560       align = left ,
561       labelsep = 0pt ,
562       label =
563         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
564     }

```

```

565 \newlist { tabularnotes* } { enumerate* } { 1 }
566 \setlist [ tabularnotes* ]
567 {
568     afterlabel = \nobreak ,
569     itemjoin = \quad ,
570     label =
571     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
572 }

```

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`.

```

573 \NewDocumentCommand \tabularnote { o m }
574 {
575     \bool_lazy_or:nnT { \cs_if_exist_p:N \@capytype } { \l_@@_in_env_bool }
576     {
577         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
578         { \@@_error:n { tabularnote~forbidden } }
579         {
580             \bool_if:NTF \l_@@_in_caption_bool
581             \@@_tabularnote_caption:nn
582             \@@_tabularnote:nn
583             { #1 } { #2 }
584         }
585     }
586 }
587 {
588     \NewDocumentCommand \tabularnote { o m }
589     {
590         \@@_error_or_warning:n { enumitem~not~loaded }
591         \@@_gredirect_none:n { enumitem~not~loaded }
592     }
593 }
594 }
595 }
596 \cs_new_protected:Npn \@@_test_first_novalued:nnn #1 #2 #3
597 { \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_novalued_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

598 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
599 {

```

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).

```

600 \int_zero:N \l_tmpa_int
601 \bool_if:NT \l_@@_notes_detect_duplicates_bool
602 {

```

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

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

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

```

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

```

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

```

641     \hbox_set:Nn \l_tmpa_box
642     {

```

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

```

643         \@@_notes_label_in_tabular:n
644         {
645             \seq_use:Nnnn
646             \l_@@_notes_labels_seq { , } { , } { , }
647         }
648     }

```

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

```

649     \int_gdecr:N \c@tabularnote
650     \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.

```

651     \int_gincr:N \g_@@_tabularnote_int
652     \refstepcounter { tabularnote }
653     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
654     { \int_gincr:N \c@tabularnote }

```

```

655     \seq_clear:N \l_@@_notes_labels_seq
656     \bool_lazy_or:nnTF
657     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
658     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
659     {
660       \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).

```

661       \skip_horizontal:n { \box_wd:N \l_tmpa_box }
662     }
663     { \box_use:N \l_tmpa_box }
664   }
665 }

```

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`.

```

666 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
667 {
668   \bool_if:NTF \g_@@_caption_finished_bool
669   {
670     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
671     { \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`!

```

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

```

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

```

676     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
677     {

```

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.

```

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

```

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!

```

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

```



```

695     \seq_clear:N \l_@@_notes_labels_seq
696   }
697 }

698 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
699 { \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.

```

700 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
701 {
702   \begin { pgfscope }
703   \pgfset
704     {
705       inner~sep = \c_zero_dim ,
706       minimum~size = \c_zero_dim
707     }
708   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
709   \pgfnode
710     { rectangle }
711     { center }
712     {
713       \vbox_to_ht:nn
714         { \dim_abs:n { #5 - #3 } }
715         {
716           \vfill
717           \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
718         }
719     }
720     { #1 }
721     { }
722   \end { pgfscope }
723 }

```

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.

```

724 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
725 {
726   \begin { pgfscope }
727   \pgfset
728     {
729       inner~sep = \c_zero_dim ,
730       minimum~size = \c_zero_dim
731     }
732   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
733   \pgfpointdiff { #3 } { #2 }
734   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
735   \pgfnode
736     { rectangle }
737     { center }
738     {
739       \vbox_to_ht:nn
740         { \dim_abs:n \l_tmpb_dim }
741         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
742     }
743     { #1 }

```

```

744     { }
745     \end { pgfscope }
746 }

```

7 The options

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

```

747 \tl_new:N \l_@@_caption_tl
748 \tl_new:N \l_@@_short_caption_tl
749 \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).

```

750 \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`.

```

751 \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`).

```

752 \dim_new:N \l_@@_cell_space_top_limit_dim
753 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

754 \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.

```

755 \dim_new:N \l_@@_xdots_inter_dim
756 \hook_gput_code:nnn { begindocument } { . }
757 { \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).

```

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

```

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.

```
765 \dim_new:N \l_@@_xdots_radius_dim
766 \hook_gput_code:nnn { begindocument } { . }
767 { \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.

```
768 \tl_new:N \l_@@_xdots_line_style_tl
769 \tl_const:Nn \c_@@_standard_tl { standard }
770 \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`.

```
771 \bool_new:N \l_@@_light_syntax_bool
772 \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).

```
773 \tl_new:N \l_@@_baseline_tl
774 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
775 \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`).

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

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

```
777 \bool_new:N \l_@@_parallelize_diags_bool
778 \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`.

```
779 \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.).

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

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

```
781 \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).

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

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

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

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

```
784 \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”.

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

```
786 \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).

```
787 \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`).

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

```
789 \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`.

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

```
791 \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.

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

```
793 \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 “:”.

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

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

```
795 \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 fonction 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.

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

```

797 \keys_define:nn { nicematrix / xdots }
798 {
799   shorten-start .code:n =
800     \hook_gput_code:nnn { begindocument } { . }
801     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
802   shorten-end .code:n =
803     \hook_gput_code:nnn { begindocument } { . }
804     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
805   shorten-start .value_required:n = true ,
806   shorten-end .value_required:n = true ,
807   shorten .code:n =
808     \hook_gput_code:nnn { begindocument } { . }
809     {
810       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
811       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
812     } ,
813   shorten .value_required:n = true ,
814   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
815   horizontal-labels .default:n = true ,
816   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
817   horizontal-label .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_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:n { 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:nm { 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:nnm { 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:nNnT { \c@iRow } = { \l_@@_last_row_int }
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}
```

We will use a version a little more efficient.

```

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

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give 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:nNnT { \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_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:nmn { 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 }

```

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

```

1402 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1403 \socket_new_plug:nmn { nicematrix / create-cell-nodes } { active }
1404 {
1405   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1406   \hbox:n
1407   {

```

```

1408     \pgfsys@markposition
1409     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1410   }
1411   #1
1412   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1413   \hbox:n
1414   {
1415     \pgfsys@markposition
1416     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1417   }
1418 }

1419 \cs_new_protected:Npn \@@_print_node_cell:
1420 {
1421   \socket_use:nn { nicematrix / siunitx-wrap }
1422   { \socket_use:nn { nicematrix / create-cell-nodes } \@@_node_cell: }
1423 }

```

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

```

1424 \cs_new_protected:Npn \@@_node_cell:
1425 {
1426   \pgfpicture
1427   \pgfsetbaseline \c_zero_dim
1428   \pgfrememberpicturepositiononpagetrue
1429   \pgfset { nicematrix / cell-node }
1430   \pgfnode
1431   { rectangle }
1432   { base }
1433   {

```

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

```

1434     \set@color
1435     \box_use:N \l_@@_cell_box
1436   }
1437   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1438   { \l_@@_pgf_node_code_tl }
1439   \str_if_empty:NF \l_@@_name_str
1440   {
1441     \pgfnodealias
1442     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1443     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1444   }
1445   \endpgfpicture
1446 }

```

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).

```
1447 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1448 {
1449   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1450   { g_@@_#2 _ lines _ tl }
1451   {
1452     \use:c { @@ _ draw _ #2 : nnn }
1453     { \int_use:N \c@iRow }
1454     { \int_use:N \c@jCol }
1455     { \exp_not:n { #3 } }
1456   }
1457 }
```

```
1458 \cs_new_protected:Npn \@@_array:n
1459 {
1460 %   \begin{macrocode}
1461   \dim_set:Nn \col@sep
1462     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1463   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1464     { \def \@halignto { } }
1465     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1466   \@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)`.

```
1467   [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1468 }
1469 \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`.

```
1470 \bool_if:nTF
1471 { \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`.

```
1472 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1473 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1474 \cs_new_protected:Npn \@@_create_row_node:
1475 {
1476   \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1477   {
1478     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1479     \@@_create_row_node_i:
1480   }
1481 }
1482 \cs_new_protected:Npn \@@_create_row_node_i:
1483 {
```

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

```

1484   \hbox
1485   {
1486     \bool_if:NT \l_@@_code_before_bool
1487     {
1488       \vtop
1489       {
1490         \skip_vertical:N 0.5\arrayrulewidth
1491         \pgfsys@markposition
1492         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1493         \skip_vertical:N -0.5\arrayrulewidth
1494       }
1495     }
1496     \pgfpicture
1497     \pgfrememberpicturepositiononpagetrue
1498     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1499     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1500     \str_if_empty:NF \l_@@_name_str
1501     {
1502       \pgfnodealias
1503       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1504       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1505     }
1506     \endpgfpicture
1507   }
1508 }

1509 \cs_new_protected:Npn \@@_in_everycr:
1510 {
1511   \bool_if:NT \c_@@_recent_array_bool
1512   {
1513     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1514     \tbl_update_cell_data_for_next_row:
1515   }
1516   \int_gzero:N \c@jCol
1517   \bool_gset_false:N \g_@@_after_col_zero_bool
1518   \bool_if:NF \g_@@_row_of_col_done_bool
1519   {
1520     \@@_create_row_node:

```

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

```

1521   \clist_if_empty:NF \l_@@_hlines_clist
1522   {
1523     \str_if_eq:eeF \l_@@_hlines_clist { all }
1524     {
1525       \clist_if_in:NeT
1526       \l_@@_hlines_clist
1527       { \int_eval:n { \c@iRow + 1 } }
1528     }
1529   }

```

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.

```

1530     \int_compare:nNnT { \c@iRow } > { -1 }
1531     {
1532       \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1533       { \hrule height \arrayrulewidth width \c_zero_dim }
1534     }
1535   }
1536 }
1537 }
1538 }

```

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

```

1539 \cs_set_protected:Npn \@@_renew_dots:
1540 {
1541   \cs_set_eq:NN \ldots \@@_Ldots:
1542   \cs_set_eq:NN \cdots \@@_Cdots:
1543   \cs_set_eq:NN \vdots \@@_Vdots:
1544   \cs_set_eq:NN \ddots \@@_Ddots:
1545   \cs_set_eq:NN \iddots \@@_Iddots:
1546   \cs_set_eq:NN \dots \@@_Ldots:
1547   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1548 }

```

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 redefinition⁵.

```

1549 \hook_gput_code:nnn { begindocument } { . }
1550 {
1551   \IfPackageLoadedTF { booktabs }
1552   {
1553     \cs_new_protected:Npn \@@_patch_booktabs:
1554     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1555   }
1556   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1557 }

```

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`.

```

1558 \cs_new_protected:Npn \@@_some_initialization:
1559 {
1560   \@@_everycr:
1561   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1562   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1563   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1564   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1565   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1566   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1567 }

```

```

1568 \cs_new_protected:Npn \@@_pre_array_ii:
1569 {

```

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

```

1570   \fp_gzero:N \g_@@_total_X_weight_fp
1571   \@@_expand_clist:N \l_@@_hlines_clist
1572   \@@_expand_clist:N \l_@@_vlines_clist
1573   \@@_patch_booktabs:

```

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

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

```

1574     \box_clear_new:N \l_@@_cell_box
1575     \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}`).

```

1576     \bool_if:NT \l_@@_small_bool
1577     {
1578         \def \arraystretch { 0.47 }
1579         \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1580         \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1581     }

```

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

```

1582     \bool_if:NT \g_@@_create_cell_nodes_bool
1583     {
1584         \tl_put_right:Nn \@@_begin_of_row:
1585         {
1586             \pgfsys@markposition
1587             { \@@_env: - row - \int_use:N \c@iRow - base }
1588         }
1589         \socket_assign_plug:nm { nicematrix / create-cell-nodes } { active }
1590     }

```

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`.

```

1591     \bool_if:nTF
1592     { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1593     {
1594         \def \ar@ialign
1595         {
1596             \bool_if:NT \c_@@_testphase_table_bool
1597             \tbl_init_cell_data_for_table:
1598             \@@_some_initialization:
1599             \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`.

```

1600         \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1601         \halign
1602     }
1603 }

```

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`.

```

1604     {
1605         \def \ialign
1606         {
1607             \@@_some_initialization:
1608             \dim_zero:N \tabskip
1609             \cs_set_eq:NN \ialign \@@_old_ialign:
1610             \halign
1611         }
1612     }

```

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).

```

1613     \bool_if:NT \c_@@_revtex_bool
1614     {
1615         \IfPackageLoadedT { colortbl }
1616         { \cs_set_protected:Npn \CT@setup { } }
1617     }

```

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).

```

1618     \cs_set_eq:NN \@@_old_ldots: \ldots
1619     \cs_set_eq:NN \@@_old_cdots: \cdots
1620     \cs_set_eq:NN \@@_old_vdots: \vdots
1621     \cs_set_eq:NN \@@_old_ddots: \ddots
1622     \cs_set_eq:NN \@@_old_iddots: \iddots
1623     \bool_if:NTF \l_@@_standard_cline_bool
1624     { \cs_set_eq:NN \cline \@@_standard_cline: }
1625     { \cs_set_eq:NN \cline \@@_cline: }
1626     \cs_set_eq:NN \Ldots \@@_Ldots:
1627     \cs_set_eq:NN \Cdots \@@_Cdots:
1628     \cs_set_eq:NN \Vdots \@@_Vdots:
1629     \cs_set_eq:NN \Ddots \@@_Ddots:
1630     \cs_set_eq:NN \Iddots \@@_Iddots:
1631     \cs_set_eq:NN \Hline \@@_Hline:
1632     \cs_set_eq:NN \Hspace \@@_Hspace:
1633     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1634     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1635     \cs_set_eq:NN \Block \@@_Block:
1636     \cs_set_eq:NN \rotate \@@_rotate:
1637     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1638     \cs_set_eq:NN \dotfill \@@_dotfill:
1639     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1640     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1641     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1642     \cs_set_eq:NN \TopRule \@@_TopRule
1643     \cs_set_eq:NN \MidRule \@@_MidRule
1644     \cs_set_eq:NN \BottomRule \@@_BottomRule
1645     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1646     \cs_set_eq:NN \Hbrace \@@_Hbrace
1647     \cs_set_eq:NN \Vbrace \@@_Vbrace
1648     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1649     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1650     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1651     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1652     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1653     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1654     \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1655     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1656     \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1657     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1658     \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:`.

```

1659     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1660     \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1661     { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1662     \@@_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!).

```

1663     \tl_if_exist:NT \l_@@_note_in_caption_tl
1664     {
1665         \tl_if_empty:NF \l_@@_note_in_caption_tl
1666         {
1667             \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1668             \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1669         }
1670     }

```

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).

```

1671     \seq_gclear:N \g_@@_multicolumn_cells_seq
1672     \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).

```

1673     \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`).

```

1674     \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.

```

1675     \int_gzero_new:N \g_@@_col_total_int
1676     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1677     \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.

```

1678     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1679     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1680     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1681     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1682     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1683     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1684     \tl_gclear:N \g_nicematrix_code_before_tl
1685     \tl_gclear:N \g_@@_pre_code_before_tl
1686 }

```

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

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

```

1687 \cs_new_protected:Npn \@@_pre_array:
1688 {
1689     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1690     \int_gzero_new:N \c@iRow
1691     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1692     \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).

```

1693   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1694   {
1695     \bool_set_true:N \l_@@_last_row_without_value_bool
1696     \bool_if:NT \g_@@_aux_found_bool
1697       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1698   }
1699   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1700   {
1701     \bool_if:NT \g_@@_aux_found_bool
1702       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1703   }

```

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”.

```

1704   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1705   {
1706     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1707       {
1708         \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1709         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1710         \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1711         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1712       }
1713   }

1714   \seq_gclear:N \g_@@_cols_vlism_seq
1715   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1716   \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.

```

1717   \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1718   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1719   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1720   \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.

```

1721   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1722   \@@_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.

```
1723 \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]`).

```
1724 \dim_zero_new:N \l_@@_left_delim_dim
1725 \dim_zero_new:N \l_@@_right_delim_dim
1726 \bool_if:NTF \g_@@_delims_bool
1727 {
```

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

```
1728 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1729 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1730 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1731 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1732 }
1733 {
1734 \dim_gset:Nn \l_@@_left_delim_dim
1735 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1736 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1737 }
```

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).

```
1738 \hbox_set:Nw \l_@@_the_array_box
1739 \skip_horizontal:N \l_@@_left_margin_dim
1740 \skip_horizontal:N \l_@@_extra_left_margin_dim
1741 \bool_if:NT \c_@@_recent_array_bool
1742 { \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).

```
1743 \m@th
1744 \c_math_toggle_token
1745 \bool_if:NTF \l_@@_light_syntax_bool
1746 { \use:c { @@-light-syntax } }
1747 { \use:c { @@-normal-syntax } }
1748 }
```

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

```
1749 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1750 {
1751 \tl_set:Nn \l_tmpa_tl { #1 }
1752 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1753 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1754 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1755 \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...

```
1756 \@@_pre_array:
1757 }
```

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).

```
1758 \cs_new_protected:Npn \@@_pre_code_before:
1759 {
```

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).

```
1760 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1761 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1762 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1763 \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.

```
1764 \pgfsys@markposition { \@@_env: - position }
1765 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1766 \pgfpicture
1767 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1768 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1769 {
1770 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1771 \pgfcoordinate { \@@_env: - row - ##1 }
1772 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1773 }
```

Now, the recreation of the `col` nodes.

```
1774 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1775 {
1776 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1777 \pgfcoordinate { \@@_env: - col - ##1 }
1778 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1779 }
```

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

```
1780 \@@_create_diag_nodes:
```

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

```
1781 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1782 \endpgfpicture
```

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

```
1783 \@@_create_blocks_nodes:
1784 \IfPackageLoadedT { tikz }
1785 {
1786 \tikzset
1787 {
1788 every-picture / .style =
1789 { overlay , name-prefix = \@@_env: - }
1790 }
1791 }
1792 \cs_set_eq:NN \cellcolor \@@_cellcolor
1793 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1794 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1795 \cs_set_eq:NN \rowcolor \@@_rowcolor
1796 \cs_set_eq:NN \rowcolors \@@_rowcolors
```

```

1797 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1798 \cs_set_eq:NN \arraycolor \@@_arraycolor
1799 \cs_set_eq:NN \columncolor \@@_columncolor
1800 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1801 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1802 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1803 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1804 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1805 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1806 }

```

```

1807 \cs_new_protected:Npn \@@_exec_code_before:
1808 {

```

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...

```

1809 \clist_map_inline:Nn \l_@@_corners_cells_clist
1810 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1811 \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`.

```

1812 \@@_add_to_colors_seq:nn { { nocolor } } { }
1813 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1814 \group_begin:

```

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

```

1815 \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`).

```

1816 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1817 { \@@_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`.

```

1818 \exp_last_unbraced:No \@@_CodeBefore_keys:
1819 \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.

```

1820 \@@_actually_color:
1821 \l_@@_code_before_tl
1822 \q_stop
1823 \bool_if:NT \l_@@_tabular_bool { \c_math_toggle_token }
1824 \group_end:
1825 }

```

```

1826 \keys_define:nm { nicematrix / CodeBefore }
1827 {
1828   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1829   create-cell-nodes .default:n = true ,
1830   sub-matrix .code:n = \keys_set:nm { nicematrix / sub-matrix } { #1 } ,
1831   sub-matrix .value_required:n = true ,

```

```

1832     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1833     delimiters / color .value_required:n = true ,
1834     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1835 }

1836 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1837 {
1838     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1839     \@@_CodeBefore:w
1840 }

```

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.

```

1841 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1842 {
1843     \bool_if:NT \g_@@_aux_found_bool
1844     {
1845         \@@_pre_code_before:
1846         \legacy_if:nF { measuring@ } { #1 }
1847     }
1848 }

```

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.

```

1849 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1850 {
1851     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1852     {
1853         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1854         \pgfcoordinate { \@@_env: - row - ##1 - base }
1855         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1856     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1857     {
1858         \cs_if_exist:cT
1859         { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1860         {
1861             \pgfsys@getposition
1862             { \@@_env: - ##1 - #####1 - NW }
1863             \@@_node_position:
1864             \pgfsys@getposition
1865             { \@@_env: - ##1 - #####1 - SE }
1866             \@@_node_position_i:
1867             \@@_pgf_rect_node:nnn
1868             { \@@_env: - ##1 - #####1 }
1869             { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1870             { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1871         }
1872     }
1873 }
1874 \@@_create_extra_nodes:
1875 \@@_create_aliases_last:
1876 }

1877 \cs_new_protected:Npn \@@_create_aliases_last:
1878 {
1879     \int_step_inline:nn { \c@iRow }
1880     {
1881         \pgfnodealias
1882         { \@@_env: - ##1 - last }
1883         { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1884 }
1885 \int_step_inline:nn { \c@jCol }
1886 {
1887   \pgfnodealias
1888     { \@@_env: - last - ##1 }
1889     { \@@_env: - \int_use:N \c@iRow - ##1 }
1890 }
1891 \pgfnodealias % added 2025-04-05
1892   { \@@_env: - last - last }
1893   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1894 }

```

```

1895 \cs_new_protected:Npn \@@_create_blocks_nodes:
1896 {
1897   \pgfpicture
1898   \pgf@relevantforpicturesizefalse
1899   \pgfrememberpicturepositiononpagetrue
1900   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1901     { \@@_create_one_block_node:nnnnn ##1 }
1902   \endpgfpicture
1903 }

```

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

```

1904 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1905 {
1906   \tl_if_empty:nF { #5 }
1907   {
1908     \@@_qpoint:n { col - #2 }
1909     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1910     \@@_qpoint:n { #1 }
1911     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1912     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1913     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1914     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1915     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1916     \@@_pgf_rect_node:nnnnn
1917       { \@@_env: - #5 }
1918       { \dim_use:N \l_tmpa_dim }
1919       { \dim_use:N \l_tmpb_dim }
1920       { \dim_use:N \l_@@_tmpc_dim }
1921       { \dim_use:N \l_@@_tmpd_dim }
1922   }
1923 }

```

```

1924 \cs_new_protected:Npn \@@_patch_for_revtext:
1925 {
1926   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1927   \cs_set_eq:NN \@array \@array@array
1928   \cs_set_eq:NN \@tabular \@tabular@array
1929   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1930   \cs_set_eq:NN \array \array@array
1931   \cs_set_eq:NN \endarray \endarray@array
1932   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1933   \cs_set_eq:NN \@mkpream \@mkpream@array
1934   \cs_set_eq:NN \@classx \@classx@array
1935   \cs_set_eq:NN \insert@column \insert@column@array
1936   \cs_set_eq:NN \@arraycr \@arraycr@array

```

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).

```

1937 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1938 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1939 }

```

10 The environment `{NiceArrayWithDelims}`

```

1940 \NewDocumentEnvironment { NiceArrayWithDelims }
1941 { m m 0 { } m ! 0 { } t \CodeBefore }
1942 {
1943 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1944 \@@_provide_pgfsyspdfmark:
1945 \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 expositant to a matrix in a mathematical formula.

```

1946 \bgroup

1947 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1948 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1949 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1950 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1951 \int_gzero:N \g_@@_block_box_int
1952 \dim_gzero:N \g_@@_width_last_col_dim
1953 \dim_gzero:N \g_@@_width_first_col_dim
1954 \bool_gset_false:N \g_@@_row_of_col_done_bool
1955 \str_if_empty:NT \g_@@_name_env_str
1956 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1957 \bool_if:NTF \l_@@_tabular_bool
1958 { \mode_leave_vertical: }
1959 { \@@_test_if_math_mode: }
1960 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in-env } }
1961 \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.

```

1962 \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.

```

1963 \cs_if_exist:NT \tikz@library@external@loaded
1964 {
1965 \tikzexternaldisable
1966 \cs_if_exist:NT \ifstandalone
1967 { \tikzset { external / optimize = false } }
1968 }

```

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

```

1969 \int_gincr:N \g_@@_env_int
1970 \bool_if:NF \l_@@_block_auto_columns_width_bool
1971 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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.

```
1972 \seq_gclear:N \g_@@_blocks_seq
1973 \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`.

```
1974 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1975 \seq_gclear:N \g_@@_pos_of_xdots_seq
1976 \tl_gclear_new:N \g_@@_code_before_tl
1977 \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.

```
1978 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1979 {
1980   \bool_gset_true:N \g_@@_aux_found_bool
1981   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1982 }
1983 { \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.

```
1984 \tl_gclear:N \g_@@_aux_tl
1985 \tl_if_empty:NF \g_@@_code_before_tl
1986 {
1987   \bool_set_true:N \l_@@_code_before_bool
1988   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1989 }
1990 \tl_if_empty:NF \g_@@_pre_code_before_tl
1991 { \bool_set_true:N \l_@@_code_before_bool }
```

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

```
1992 \bool_if:NTF \g_@@_delims_bool
1993 { \keys_set:nn { nicematrix / pNiceArray } }
1994 { \keys_set:nn { nicematrix / NiceArray } }
1995 { #3 , #5 }

1996 \@@_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:.`

```
1997 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1998 }
```

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

```
1999 {
2000   \bool_if:NTF \l_@@_light_syntax_bool
2001   { \use:c { end @@-light-syntax } }
2002   { \use:c { end @@-normal-syntax } }
2003   \c_math_toggle_token
2004   \skip_horizontal:N \l_@@_right_margin_dim
2005   \skip_horizontal:N \l_@@_extra_right_margin_dim
2006
2007   % awful workaround
2008   \int_if_zero:nT { \g_@@_col_total_int }
2009   {
```



```

2010     \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim }
2011     {
2012         \skip_horizontal:n { - \l_@@_columns_width_dim }
2013         \bool_if:NTF \l_@@_tabular_bool
2014         { \skip_horizontal:n { - 2 \tabcolsep } }
2015         { \skip_horizontal:n { - 2 \arraycolsep } }
2016     }
2017 }
2018 \hbox_set_end:
2019 \bool_if:NT \c_@@_recent_array_bool
2020 { \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.

```

2021     \bool_if:NT \l_@@_width_used_bool
2022     {
2023         \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2024         { \@@_error_or_warning:n { width-without-X-columns } }
2025     }

```

Now, if there is at least one `X`-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `\l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a `X`-column of weight x , the width will be `\l_@@_X_columns_dim` multiplied by x .

```

2026     \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2027     { \@@_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).

```

2028     \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2029     {
2030         \bool_if:NF \l_@@_last_row_without_value_bool
2031         {
2032             \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2033             {
2034                 \@@_error:n { Wrong-last-row }
2035                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2036             }
2037         }
2038     }

```

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

```

2039     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2040     \bool_if:NTF \g_@@_last_col_found_bool
2041     { \int_gdecr:N \c@jCol }
2042     {
2043         \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2044         { \@@_error:n { last-col-not-used } }
2045     }

```

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

```

2046     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2047     \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2048     { \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).

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

```

2049 \int_if_zero:nT { \l_@@_first_col_int }
2050 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2051 \bool_if:nTF { ! \g_@@_delims_bool }
2052 {
2053   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2054   { \@@_use_arraybox_with_notes_c: }
2055   {
2056     \str_if_eq:eeTF \l_@@_baseline_tl { b }
2057     { \@@_use_arraybox_with_notes_b: }
2058     { \@@_use_arraybox_with_notes: }
2059   }
2060 }

```

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).

```

2061 {
2062   \int_if_zero:nTF { \l_@@_first_row_int }
2063   {
2064     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2065     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2066   }
2067   { \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

```

2068 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2069 {
2070   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2071   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2072 }
2073 { \dim_zero:N \l_tmpb_dim }
2074 \hbox_set:Nn \l_tmpa_box
2075 {
2076   \m@th
2077   \c_math_toggle_token
2078   \@@_color:o \l_@@_delimiters_color_tl
2079   \exp_after:wN \left \g_@@_left_delim_tl
2080   \vcenter
2081   {

```

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.

```

2082   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2083   \hbox
2084   {
2085     \bool_if:NTF \l_@@_tabular_bool
2086     { \skip_horizontal:n { - \tabcolsep } }
2087     { \skip_horizontal:n { - \arraycolsep } }
2088     \@@_use_arraybox_with_notes_c:
2089     \bool_if:NTF \l_@@_tabular_bool
2090     { \skip_horizontal:n { - \tabcolsep } }
2091     { \skip_horizontal:n { - \arraycolsep } }
2092   }

```

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

```

2093   \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2094 }
2095 \exp_after:wN \right \g_@@_right_delim_tl
2096 \c_math_toggle_token
2097 }

```

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).

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.

```

2098     \bool_if:NTF \l_@@_delimiters_max_width_bool
2099     {
2100         \@@_put_box_in_flow_bis:nn
2101         { \g_@@_left_delim_tl }
2102         { \g_@@_right_delim_tl }
2103     }
2104     \@@_put_box_in_flow:
2105 }

```

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).

```

2106     \bool_if:NT \g_@@_last_col_found_bool
2107     { \skip_horizontal:N \g_@@_width_last_col_dim }
2108     \bool_if:NT \l_@@_preamble_bool
2109     {
2110         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2111         { \@@_warning_gredirect_none:n { columns-not-used } }
2112     }
2113     \@@_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.

```

2114     \egroup

```

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

```

2115     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2116     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2117     \iow_now:Ne \@mainaux
2118     {
2119         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2120         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2121         { \exp_not:o \g_@@_aux_tl }
2122     }
2123     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2124     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2125 }

```

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

```

2126     \cs_new_protected:Npn \@@_compute_width_X:
2127     {
2128         \tl_gput_right:Ne \g_@@_aux_tl
2129         {
2130             \bool_set_true:N \l_@@_X_columns_aux_bool
2131             \dim_set:Nn \l_@@_X_columns_dim
2132             {
2133                 \dim_compare:nNnTF
2134                 {
2135                     \dim_abs:n
2136                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2137                 }
2138                 <
2139                 { 0.001 pt }
2140                 { \dim_use:N \l_@@_X_columns_dim }
2141                 {
2142                     \dim_eval:n

```

```

2143     {
2144         \fp_to_dim:n
2145         {
2146             (
2147                 \dim_eval:n
2148                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2149             )
2150             / \fp_use:N \g_@@_total_X_weight_fp
2151         }
2152         + \l_@@_X_columns_dim
2153     }
2154 }
2155 }
2156 }
2157 }

```

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.

```

2158 \cs_new_protected:Npn \@@_transform_preamble:
2159 {
2160     \@@_transform_preamble_i:
2161     \@@_transform_preamble_ii:
2162 }
2163 \cs_new_protected:Npn \@@_transform_preamble_i:
2164 {
2165     \int_gzero:N \c@jCol

```

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

```

2166     \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.

```

2167     \bool_gset_false:N \g_tmpb_bool

```

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

```

2168     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2169     \int_zero:N \l_tmpa_int
2170     \tl_gclear:N \g_@@_array_preamble_tl
2171     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2172     {
2173         \tl_gset:Nn \g_@@_array_preamble_tl
2174         { ! { \skip_horizontal:N \arrayrulewidth } }
2175     }
2176     {
2177         \clist_if_in:NnT \l_@@_vlines_clist 1
2178         {
2179             \tl_gset:Nn \g_@@_array_preamble_tl
2180             { ! { \skip_horizontal:N \arrayrulewidth } }
2181         }
2182     }

```

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

```

2183 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2184 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2185 \@@_replace_columncolor:
2186 }

2187 \cs_new_protected:Npn \@@_transform_preamble_ii:
2188 {

```

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

```

2189 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2190 {
2191   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2192   { \bool_gset_true:N \g_@@_delims_bool }
2193 }
2194 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2195 \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).

```

2196 \int_if_zero:nTF { \l_@@_first_col_int }
2197 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2198 {
2199   \bool_if:NF \g_@@_delims_bool
2200   {
2201     \bool_if:NF \l_@@_tabular_bool
2202     {
2203       \clist_if_empty:NT \l_@@_vlines_clist
2204       {
2205         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2206         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2207       }
2208     }
2209   }
2210 }
2211 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2212 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2213 {
2214   \bool_if:NF \g_@@_delims_bool
2215   {
2216     \bool_if:NF \l_@@_tabular_bool
2217     {
2218       \clist_if_empty:NT \l_@@_vlines_clist
2219       {
2220         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2221         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2222       }
2223     }
2224   }
2225 }

```

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`).

```

2226 \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
2227 {

```

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.

```

2228     \bool_if:NF \c_@@_testphase_table_bool
2229     {
2230         \tl_gput_right:Nn \g_@@_array_preamble_tl
2231         { > { \@@_error_too_much_cols: } l }
2232     }
2233 }
2234 }

```

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.

```

2235 \cs_new_protected:Npn \@@_rec_preamble:n #1
2236 {

```

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

```

2237     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2238     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2239     {

```

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

```

2240     \cs_if_exist:cTF { NC @ find @ #1 }
2241     {
2242         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2243         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2244     }
2245     {
2246         \str_if_eq:nnTF { #1 } { S }
2247         { \@@_fatal:n { unknown~column~type~S } }
2248         { \@@_fatal:nn { unknown~column~type } { #1 } }
2249     }
2250 }
2251 }

```

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

```

2252 \cs_new_protected:Npn \@@_c: #1
2253 {
2254     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2255     \tl_gclear:N \g_@@_pre_cell_tl
2256     \tl_gput_right:Nn \g_@@_array_preamble_tl
2257     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2258     \int_gincr:N \c@jCol
2259     \@@_rec_preamble_after_col:n
2260 }

2261 \cs_new_protected:Npn \@@_l: #1
2262 {
2263     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2264     \tl_gclear:N \g_@@_pre_cell_tl
2265     \tl_gput_right:Nn \g_@@_array_preamble_tl
2266     {
2267         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2268         l
2269         < \@@_cell_end:

```

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`.

```

2270     }
2271     \int_gincr:N \c@jCol
2272     \@@_rec_preamble_after_col:n
2273 }

2274 \cs_new_protected:Npn \@@_r: #1
2275 {
2276     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2277     \tl_gclear:N \g_@@_pre_cell_tl
2278     \tl_gput_right:Nn \g_@@_array_preamble_tl
2279     {
2280         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2281         r
2282         < \@@_cell_end:
2283     }
2284     \int_gincr:N \c@jCol
2285     \@@_rec_preamble_after_col:n
2286 }

```

For ! and @

```

2287 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2288 {
2289     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2290     \@@_rec_preamble:n
2291 }
2292 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2293 \cs_new_protected:cpn { @@ _ | : } #1
2294 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2295     \int_incr:N \l_tmpa_int
2296     \@@_make_preamble_i_i:n
2297 }

2298 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2299 {
2300     \str_if_eq:nnTF { #1 } { | }
2301     { \use:c { @@ _ | : } | }
2302     { \@@_make_preamble_i_ii:nn { } #1 }
2303 }

2304 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2305 {
2306     \str_if_eq:nnTF { #2 } { [ ] }
2307     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2308     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2309 }

2310 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2311 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2312 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2313 {
2314     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2315     \tl_gput_right:Ne \g_@@_array_preamble_tl
2316     {

```

Here, the command \dim_use:N is mandatory.

```

2317         \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2318     }
2319     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2320     {
2321         \@@_vline:n
2322         {
2323             position = \int_eval:n { \c@jCol + 1 } ,

```

```

2324     multiplicity = \int_use:N \l_tmpa_int ,
2325     total-width = \dim_use:N \l_@@_rule_width_dim ,
2326     #2
2327 }

```

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.

```

2328     }
2329     \int_zero:N \l_tmpa_int
2330     \str_if_eq:nnF { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2331     \@@_rec_preamble:n #1
2332 }

2333 \cs_new_protected:cpn { @@_ > : } #1 #2
2334 {
2335     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2336     \@@_rec_preamble:n
2337 }

2338 \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.

```

2339 \keys_define:nn { nicematrix / p-column }
2340 {
2341     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2342     r .value_forbidden:n = true ,
2343     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2344     c .value_forbidden:n = true ,
2345     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2346     l .value_forbidden:n = true ,
2347     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2348     S .value_forbidden:n = true ,
2349     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2350     p .value_forbidden:n = true ,
2351     t .meta:n = p ,
2352     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2353     m .value_forbidden:n = true ,
2354     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2355     b .value_forbidden:n = true
2356 }

```

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

```

2357 \cs_new_protected:Npn \@@_p: #1
2358 {
2359     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2360     \@@_make_preamble_ii_i:n
2361 }
2362 \cs_set_eq:NN \@@_b: \@@_p:
2363 \cs_set_eq:NN \@@_m: \@@_p:

2364 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2365 {
2366     \str_if_eq:nnTF { #1 } { [ ]
2367         { \@@_make_preamble_ii_ii:w [ ]
2368             { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2369         }
2370     \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2371     { \@@_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.

```
2372 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2373 {
```

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).

```
2374   \str_set:Nn \l_@@_hpos_col_str { j }
2375   \@@_keys_p_column:n { #1 }
2376   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2377 }

2378 \cs_new_protected:Npn \@@_keys_p_column:n #1
2379 { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }
```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`. The third is some code added at the beginning of the cell.

```
2380 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2381 {
2382   \use:e
2383   {
2384     \@@_make_preamble_ii_v:nnnnnnnn
2385     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2386     { \dim_eval:n { #1 } }
2387   }
```

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.

```
2388   \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2389   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2390   {
```

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

```
2391       \def \exp_not:N \l_@@_hpos_cell_tl
2392       { \str_lowercase:f { \l_@@_hpos_col_str } }
2393   }
2394   \IfPackageLoadedTF { ragged2e }
2395   {
2396     \str_case:on \l_@@_hpos_col_str
2397     {
```

The following `\exp_not:N` are mandatory.

```
2398         c { \exp_not:N \Centering }
2399         l { \exp_not:N \RaggedRight }
2400         r { \exp_not:N \RaggedLeft }
2401     }
2402   }
2403   {
2404     \str_case:on \l_@@_hpos_col_str
2405     {
2406       c { \exp_not:N \centering }
2407       l { \exp_not:N \raggedright }
2408       r { \exp_not:N \raggedleft }
2409     }
2410   }
2411   #3
2412 }
2413 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2414 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2415 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2416 { #2 }
2417 {
2418   \str_case:onF \l_@@_hpos_col_str
```

```

2419         {
2420             { j } { c }
2421             { si } { c }
2422         }

```

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

```

2423             { \str_lowercase:f \l_@@_hpos_col_str }
2424         }
2425     }

```

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

```

2426     \int_gincr:N \c@jCol
2427     \@@_rec_preamble_after_col:n
2428 }

```

#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*.

```

2429 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2430 {
2431     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2432     {
2433         \tl_gput_right:Nn \g_@@_array_preamble_tl
2434         { > \@@_test_if_empty_for_S: }
2435     }
2436     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2437     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2438     \tl_gclear:N \g_@@_pre_cell_tl
2439     \tl_gput_right:Nn \g_@@_array_preamble_tl
2440     {
2441         > {

```

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.

```

2442         \dim_set:Nn \l_@@_col_width_dim { #2 }
2443         \bool_if:NT \c_@@_testphase_table_bool
2444         { \tag_struct_begin:n { tag = Div } }
2445         \@@_cell_begin:

```

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

```

2446         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2447     \everypar
2448     {
2449         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2450         \everypar { }
2451     }
2452     \bool_if:NT \c_@@_testphase_table_bool { \tagpdfparaOn }

```

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

```

2453         #3

```

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

```

2454     \g_@@_row_style_tl
2455     \arraybackslash
2456     #5
2457   }
2458   #8
2459   < {
2460     #6

```

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

```

2461     \@finalstrut \@arstrutbox
2462     \use:c { end #7 }

```

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

```

2463     #4
2464     \@@_cell_end:
2465     \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2466   }
2467 }
2468 }

```

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

```

2469 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2470 {

```

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 `&`).

```

2471   \group_align_safe_begin:
2472   \peek_meaning:NTF &
2473   { \@@_the_cell_is_empty: }
2474   {
2475     \peek_meaning:NTF \\\
2476     { \@@_the_cell_is_empty: }
2477     {
2478       \peek_meaning:NTF \crcr
2479       \@@_the_cell_is_empty:
2480       \group_align_safe_end:
2481     }
2482   }
2483 }
2484 \cs_new_protected:Npn \@@_the_cell_is_empty:
2485 {
2486   \group_align_safe_end:
2487   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488   {

```

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

```

2489     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2490     \skip_horizontal:N \l_@@_col_width_dim
2491   }
2492 }
2493 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2494 {
2495   \peek_meaning:NT \__siunitx_table_skip:n
2496   { \bool_gset_true:N \g_@@_empty_cell_bool }
2497 }

```

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.

```
2498 \cs_new_protected:Npn \@@_center_cell_box:
2499 {
```

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

```
2500 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2501 {
2502   \dim_compare:nNnT
2503     { \box_ht:N \l_@@_cell_box }
2504     >
```

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).

```
2505     { \box_ht:N \strutbox }
2506     {
2507       \hbox_set:Nn \l_@@_cell_box
2508         {
2509           \box_move_down:nn
2510             {
2511               ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2512                 + \baselineskip ) / 2
2513             }
2514           { \box_use:N \l_@@_cell_box }
2515         }
2516       }
2517     }
2518 }
```

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

```
2519 \cs_new_protected:Npn \@@_V: #1 #2
2520 {
2521   \str_if_eq:nnTF { #1 } { [ ]
2522     { \@@_make_preamble_V_i:w [ ]
2523       { \@@_make_preamble_V_i:w [ ] { #2 } }
2524     }
2525   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2526     { \@@_make_preamble_V_ii:nn { #1 } }
2527   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2528     {
2529       \str_set:Nn \l_@@_vpos_col_str { p }
2530       \str_set:Nn \l_@@_hpos_col_str { j }
2531       \@@_keys_p_column:n { #1 }
2532       \IfPackageLoadedTF { varwidth }
2533         { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2534         {
2535           \@@_error_or_warning:n { varwidth-not-loaded }
2536           \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2537         }
2538     }
```

For `w` and `W`

```
2539 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2540 \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.

```

2541 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2542 {
2543   \str_if_eq:nnTF { #3 } { s }
2544     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2545     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2546 }

```

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.

```

2547 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2548 {
2549   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2550   \tl_gclear:N \g_@@_pre_cell_tl
2551   \tl_gput_right:Nn \g_@@_array_preamble_tl
2552     {
2553       > {
2554         \dim_set:Nn \l_@@_col_width_dim { #2 }
2555         \@@_cell_begin:
2556         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2557       }
2558       c
2559       < {
2560         \@@_cell_end_for_w_s:
2561         #1
2562         \@@_adjust_size_box:
2563         \box_use_drop:N \l_@@_cell_box
2564       }
2565     }
2566   \int_gincr:N \c@jCol
2567   \@@_rec_preamble_after_col:n
2568 }

```

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

```

2569 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2570 {
2571   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2572   \tl_gclear:N \g_@@_pre_cell_tl
2573   \tl_gput_right:Nn \g_@@_array_preamble_tl
2574     {
2575       > {

```

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.

```

2576         \dim_set:Nn \l_@@_col_width_dim { #4 }
2577         \hbox_set:Nw \l_@@_cell_box
2578         \@@_cell_begin:
2579         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2580       }
2581       c
2582       < {
2583         \@@_cell_end:
2584         \hbox_set_end:
2585         #1
2586         \@@_adjust_size_box:
2587         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2588       }
2589     }

```

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

```

2590     \int_gincr:N \c@jCol
2591     \@@_rec_preamble_after_col:n
2592   }

2593 \cs_new_protected:Npn \@@_special_W:
2594   {
2595     \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2596     { \@@_warning:n { W~warning } }
2597   }

```

For S (of siunitx).

```

2598 \cs_new_protected:Npn \@@_S: #1 #2
2599   {
2600     \str_if_eq:nnTF { #2 } { [ ]
2601       { \@@_make_preamble_S:w [ ] }
2602       { \@@_make_preamble_S:w [ ] { #2 } }
2603     }

2604 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2605   { \@@_make_preamble_S_i:n { #1 } }

2606 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2607   {
2608     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2609     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2610     \tl_gclear:N \g_@@_pre_cell_tl
2611     \tl_gput_right:Nn \g_@@_array_preamble_tl
2612     {
2613       > {

```

In the cells of a column of type S, we have to wrap the command `\@@_node_cell:` for the horizontal alignment of the content of the cell (siunitx has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignment once again).

```

2614     \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2615     \keys_set:nn { siunitx } { #1 }
2616     \@@_cell_begin:
2617     \siunitx_cell_begin:w
2618   }
2619   c
2620   <
2621   {
2622     \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`).

```

2623     \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2624     {
2625       \bool_if:NTF \l__siunitx_table_text_bool
2626       { \bool_set_true:N }
2627       { \bool_set_false:N }
2628       \l__siunitx_table_text_bool
2629     }
2630     \@@_cell_end:
2631   }
2632 }

```

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

```

2633     \int_gincr:N \c@jCol
2634     \@@_rec_preamble_after_col:n
2635   }

```

For (, [and \{.

```
2636 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2637 {
2638   \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.

```
2639   \int_if_zero:nTF { \c@jCol }
2640   {
2641     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2642     {
```

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

```
2643       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2644       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2645       \@@_rec_preamble:n #2
2646     }
2647     {
2648       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2649       \@@_make_preamble_iv:nn { #1 } { #2 }
2650     }
2651   }
2652   { \@@_make_preamble_iv:nn { #1 } { #2 } }
2653 }
2654 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2655 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2656 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2657 {
2658   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2659   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2660   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2661   {
2662     \@@_error:nn { delimiter~after~opening } { #2 }
2663     \@@_rec_preamble:n
2664   }
2665   { \@@_rec_preamble:n #2 }
2666 }
```

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

```
2667 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2668 { \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}).

```
2669 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2670 {
2671   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2672   \tl_if_in:nnTF { ) ] \} } { #2 }
2673   { \@@_make_preamble_v:nnn #1 #2 }
2674   {
2675     \str_if_eq:nnTF { \s_stop } { #2 }
2676     {
2677       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2678       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2679       {
2680         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2681         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2682         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2683         \@@_rec_preamble:n #2
2684       }
2685     }
2686   }
```

```

2686     {
2687     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2688     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2689     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2690     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2691     \@@_rec_preamble:n #2
2692     }
2693   }
2694 }
2695 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2696 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2697 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2698   {
2699   \str_if_eq:nnTF { \s_stop } { #3 }
2700     {
2701     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2702     {
2703     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2704     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2705     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2706     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2707     }
2708     {
2709     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2710     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2711     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2712     \@@_error:nn { double~closing~delimiter } { #2 }
2713     }
2714   }
2715   {
2716   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2717   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2718   \@@_error:nn { double~closing~delimiter } { #2 }
2719   \@@_rec_preamble:n #3
2720   }
2721 }

2722 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2723   { \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 @{...}.

```

2724 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2725   {
2726   \str_if_eq:nnTF { #1 } { < }
2727     { \@@_rec_preamble_after_col_i:n }
2728     {
2729     \str_if_eq:nnTF { #1 } { @ }
2730     { \@@_rec_preamble_after_col_ii:n }
2731     {
2732     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2733     {
2734     \tl_gput_right:Nn \g_@@_array_preamble_tl
2735     { ! { \skip_horizontal:N \arrayrulewidth } }
2736     }
2737     {
2738     \clist_if_in:NeT \l_@@_vlines_clist
2739     { \int_eval:n { \c@jCol + 1 } }
2740     {
2741     \tl_gput_right:Nn \g_@@_array_preamble_tl
2742     { ! { \skip_horizontal:N \arrayrulewidth } }

```



```

2743     }
2744     }
2745     \@@_rec_preamble:n { #1 }
2746   }
2747 }
2748 }
2749 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2750 {
2751   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2752   \@@_rec_preamble_after_col:n
2753 }

```

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.

```

2754 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2755 {
2756   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2757   {
2758     \tl_gput_right:Nn \g_@@_array_preamble_tl
2759     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2760   }
2761   {
2762     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2763     {
2764       \tl_gput_right:Nn \g_@@_array_preamble_tl
2765       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2766     }
2767     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2768   }
2769   \@@_rec_preamble:n
2770 }
2771 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2772 {
2773   \tl_clear:N \l_tmpa_tl
2774   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2775   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2776 }

```

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

```

2777 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2778 { \@@_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.

```

2779 \cs_new_protected:Npn \@@_X: #1 #2
2780 {
2781   \str_if_eq:nnTF { #2 } { [ ]
2782     { \@@_make_preamble_X:w [ ] }
2783     { \@@_make_preamble_X:w [ ] #2 }
2784   }
2785   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2786   { \@@_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 which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```

2787 \keys_define:nm { nicematrix / X-column }
2788 {
2789   unknown .code:n =
2790     \regex_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2791     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2792     { \@@_error_or_warning:n { invalid-weight } }
2793 }

```

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

```

2794 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2795 {

```

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).

```

2796   \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).

```

2797   \str_set:Nn \l_@@_vpos_col_str { p }

```

We will store in `\l_tmpa_fp` the weight of the column (`\l_tmpa_fp` also appears in `{nicematrix/X-column}`) and the error message `invalid~weight`.

```

2798   \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

2799   \@@_keys_p_column:n { #1 }

```

The unknown keys have been stored by `\@@_keys_p_column:n` in `\l_tmpa_tl` and we use them right now in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitly provided by the final user.

```

2800   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2801   \fp_gadd:Nn \g_@@_total_X_weight_fp \l_tmpa_fp

```

We test whether we know the actual width of the X-columns by reading the `aux` file (after the first compilation, the width of the X-columns is computed and written in the `aux` file).

```

2802   \bool_if:NTF \l_@@_X_columns_aux_bool
2803   {
2804     \@@_make_preamble_ii_iv:nnn

```

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

```

2805     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2806     { minipage }
2807     { \@@_no_update_width: }
2808   }

```

In the current compilation, we don't know the actual width of the X column. However, you have to construct the cells of that column! By convention, we have decided to compose in a `{minipage}` of width 5 cm even though we will nullify `\l_@@_cell_box` after its composition.

```

2809   {
2810     \tl_gput_right:Nn \g_@@_array_preamble_tl
2811     {
2812       > {
2813         \@@_cell_begin:
2814         \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`.

```

2815         \NotEmpty

```

The following code will nullify the box of the cell.

```

2816         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2817         { \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`.

```

2818         \begin { minipage } { 5 cm } \arraybackslash
2819     }
2820     c
2821     < {
2822         \end { minipage }
2823         \@@_cell_end:
2824     }
2825 }
2826 \int_gincr:N \c@jCol
2827 \@@_rec_preamble_after_col:n
2828 }
2829 }

2830 \cs_new_protected:Npn \@@_no_update_width:
2831 {
2832     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2833     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2834 }

```

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

```

2835 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2836 {
2837     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2838     { \int_eval:n { \c@jCol + 1 } }
2839     \tl_gput_right:Ne \g_@@_array_preamble_tl
2840     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2841     \@@_rec_preamble:n
2842 }

```

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.

```

2843 \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).

```

2844 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2845 { \@@_fatal:n { Preamble-forgotten } }
2846 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2847 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2848 { @@ _ \token_to_str:N \hline : }
2849 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2851 { @@ _ \token_to_str:N \hline : }
2852 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2853 { @@ _ \token_to_str:N \hline : }
2854 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2855 { @@ _ \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).

```

2856 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2857 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2858     \multispan { #1 }
2859     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2860     \begingroup
2861     \bool_if:NT \c_@@_testphase_table_bool
2862       { \tbl_update_multicolumn_cell_data:n { #1 } }
2863     \def \@addamp
2864       { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2865     \tl_gclear:N \g_@@_preamble_tl
2866     \@@_make_m_preamble:n #2 \q_stop

```

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

```

2867     \exp_args:No \@mkpream \g_@@_preamble_tl
2868     \@addtopreamble \@empty
2869     \endgroup
2870     \bool_if:NT \c_@@_recent_array_bool
2871       { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2872     \int_compare:nNnT { #1 } > { \c_one_int }
2873     {
2874       \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2875         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2876       \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2877       \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2878         {
2879           {
2880             \int_if_zero:nTF { \c@jCol }
2881               { \int_eval:n { \c@iRow + 1 } }
2882               { \int_use:N \c@iRow }
2883           }
2884           { \int_eval:n { \c@jCol + 1 } }
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 } }

```

The last argument is for the name of the block

```

2891         { }
2892       }
2893     }

```

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

```

2894     \RenewDocumentCommand \cellcolor { 0 { } m }
2895     {
2896       \tl_gput_right:Ne \g_@@_pre_code_before_tl
2897       {
2898         \@@_rectanglecolor [ ##1 ]
2899           { \exp_not:n { ##2 } }
2900           { \int_use:N \c@iRow - \int_use:N \c@jCol }
2901           { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2902       }
2903     \ignorespaces
2904   }

```

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

```

2905 \def \@sharp { #3 }
2906 \@arstrut
2907 \@preamble
2908 \null

```

We add some lines.

```

2909 \int_gadd:Nn \c@jCol { #1 - 1 }
2910 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
2911 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2912 \ignorespaces
2913 }

```

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`.

```

2914 \cs_new_protected:Npn \@_make_m_preamble:n #1
2915 {
2916   \str_case:nnF { #1 }
2917   {
2918     c { \@_make_m_preamble_i:n #1 }
2919     l { \@_make_m_preamble_i:n #1 }
2920     r { \@_make_m_preamble_i:n #1 }
2921     > { \@_make_m_preamble_ii:nn #1 }
2922     ! { \@_make_m_preamble_ii:nn #1 }
2923     @ { \@_make_m_preamble_ii:nn #1 }
2924     | { \@_make_m_preamble_iii:n #1 }
2925     p { \@_make_m_preamble_iv:nnn t #1 }
2926     m { \@_make_m_preamble_iv:nnn c #1 }
2927     b { \@_make_m_preamble_iv:nnn b #1 }
2928     w { \@_make_m_preamble_v:nnnn { } #1 }
2929     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2930     \q_stop { }
2931   }
2932   {
2933     \cs_if_exist:cTF { NC @ find @ #1 }
2934     {
2935       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2936       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2937     }
2938     {
2939       \str_if_eq:nnTF { #1 } { S }
2940       { \@_fatal:n { unknown~column~type~S } }
2941       { \@_fatal:nn { unknown~column~type } { #1 } }
2942     }
2943   }
2944 }

```

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

```

2945 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2946 {
2947   \tl_gput_right:Nn \g_@@_preamble_tl
2948   {
2949     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2950     #1
2951     < \@_cell_end:
2952   }

```

We test for the presence of a `<`.

```

2953 \@_make_m_preamble_x:n
2954 }

```

For >, ! and @

```
2955 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2956 {
2957   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2958   \@@_make_m_preamble:n
2959 }
```

For |

```
2960 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2961 {
2962   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2963   \@@_make_m_preamble:n
2964 }
```

For p, m and b

```
2965 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2966 {
2967   \tl_gput_right:Nn \g_@@_preamble_tl
2968   {
2969     > {
2970       \@@_cell_begin:
2971       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2972       \mode_leave_vertical:
2973       \arraybackslash
2974       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2975     }
2976     c
2977     < {
2978       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2979       \end { minipage }
2980       \@@_cell_end:
2981     }
2982   }
```

We test for the presence of a <.

```
2983   \@@_make_m_preamble_x:n
2984 }
```

For w and W

```
2985 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2986 {
2987   \tl_gput_right:Nn \g_@@_preamble_tl
2988   {
2989     > {
2990       \dim_set:Nn \l_@@_col_width_dim { #4 }
2991       \hbox_set:Nw \l_@@_cell_box
2992       \@@_cell_begin:
2993       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2994     }
2995     c
2996     < {
2997       \@@_cell_end:
2998       \hbox_set_end:
2999       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3000       #1
3001       \@@_adjust_size_box:
3002       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3003     }
3004   }
```

We test for the presence of a <.

```
3005   \@@_make_m_preamble_x:n
3006 }
```

After a specifier of column, we have to test whether there is one or several $\langle \{ \dots \}$.

```

3007 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3008 {
3009   \str_if_eq:nnTF { #1 } { < }
3010     { \@@_make_m_preamble_ix:n }
3011     { \@@_make_m_preamble:n { #1 } }
3012 }
3013 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3014 {
3015   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3016   \@@_make_m_preamble_x:n
3017 }

```

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`).

```

3018 \cs_new_protected:Npn \@@_put_box_in_flow:
3019 {
3020   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3021   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3022   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3023     { \box_use_drop:N \l_tmpa_box }
3024     { \@@_put_box_in_flow_i: }
3025 }

```

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

```

3026 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3027 {
3028   \pgfpicture
3029     \@@_qpoint:n { row - 1 }
3030     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3031     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3032     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3033     \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).

```

3034   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3035     {
3036       \int_set:Nn \l_tmpa_int
3037         {
3038           \str_range:Nnn
3039             \l_@@_baseline_tl
3040             6
3041             { \tl_count:o \l_@@_baseline_tl }
3042         }
3043       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3044     }
3045     {
3046       \str_if_eq:eeTF \l_@@_baseline_tl { t }
3047         { \int_set_eq:NN \l_tmpa_int \c_one_int }
3048         {
3049           \str_if_eq:onTF \l_@@_baseline_tl { b }
3050             { \int_set_eq:NN \l_tmpa_int \c@iRow }
3051             { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3052         }
3053       \bool_lazy_or:nnT
3054         { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3055         { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }

```

```

3056     {
3057         \@@_error:n { bad-value-for-baseline }
3058         \int_set_eq:NN \l_tmpa_int \c_one_int
3059     }
3060     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3061     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3062     }
3063     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3064     \endpgfpicture
3065     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3066     \box_use_drop:N \l_tmpa_box
3067 }

```

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).

```

3068 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3069 {

```

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.

```

3070     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3071     {
3072         \int_compare:nNnT { \c_jCol } > { \c_one_int }
3073         {
3074             \box_set_wd:Nn \l_@@_the_array_box
3075             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3076         }
3077     }

```

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

```

3078     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3079     \bool_if:NT \l_@@_caption_above_bool
3080     {
3081         \tl_if_empty:NF \l_@@_caption_tl
3082         {
3083             \bool_set_false:N \g_@@_caption_finished_bool
3084             \int_gzero:N \c@tabularnote
3085             \@@_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).

```

3086         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3087         {
3088             \tl_gput_right:Ne \g_@@_aux_tl
3089             {
3090                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3091                 { \int_use:N \g_@@_notes_caption_int }
3092             }
3093             \int_gzero:N \g_@@_notes_caption_int
3094         }
3095     }
3096 }

```


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

```

3097   \hbox
3098   {
3099   \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.

```

3100   \@@_create_extra_nodes:
3101   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3102   }

```

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).

```

3103   \bool_lazy_any:nT
3104   {
3105   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3106   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3107   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3108   }
3109   \@@_insert_tabularnotes:
3110   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3111   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3112   \end { minipage }
3113   }

```

```

3114 \cs_new_protected:Npn \@@_insert_caption:
3115 {
3116   \tl_if_empty:NF \l_@@_caption_tl
3117   {
3118     \cs_if_exist:NTF \@capttype
3119     { \@@_insert_caption_i: }
3120     { \@@_error:n { caption~outside~float } }
3121   }
3122   }

```

```

3123 \cs_new_protected:Npn \@@_insert_caption_i:
3124 {
3125   \group_begin:

```

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

```

3126   \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.

```

3127   \IfPackageLoadedT { floatrow }
3128   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3129   \tl_if_empty:NTF \l_@@_short_caption_tl
3130   { \caption }
3131   { \caption [ \l_@@_short_caption_tl ] }
3132   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3133 \bool_if:NF \g_@@_caption_finished_bool
3134 {
3135   \bool_gset_true:N \g_@@_caption_finished_bool
3136   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3137   \int_gzero:N \c@tabularnote
3138 }
3139 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3140 \group_end:
3141 }
3142 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3143 {
3144   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3145   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3146 }
3147 \cs_new_protected:Npn \@@_insert_tabularnotes:
3148 {
3149   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3150   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3151   \skip_vertical:N 0.65ex

```

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

```

3152 \group_begin:
3153 \l_@@_notes_code_before_tl
3154 \tl_if_empty:NF \g_@@_tabularnote_tl
3155 {
3156   \g_@@_tabularnote_tl \par
3157   \tl_gclear:N \g_@@_tabularnote_tl
3158 }

```

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.

```

3159 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3160 {
3161   \bool_if:NTF \l_@@_notes_para_bool
3162   {
3163     \begin { tabularnotes* }
3164       \seq_map_inline:Nn \g_@@_notes_seq
3165         { \@@_one_tabularnote:nn ##1 }
3166       \strut
3167     \end { tabularnotes* }

```

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

```

3168   \par
3169 }
3170 {
3171   \tabularnotes
3172     \seq_map_inline:Nn \g_@@_notes_seq
3173       { \@@_one_tabularnote:nn ##1 }
3174     \strut
3175   \endtabularnotes
3176 }
3177 }
3178 \unskip
3179 \group_end:
3180 \bool_if:NT \l_@@_notes_bottomrule_bool
3181 {
3182   \IfPackageLoadedTF { booktabs }
3183   {

```

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

```

3184     \skip_vertical:N \aboverulesep

```

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

```

3185         { \CT@arc@ \hrule height \heavyrulewidth }
3186     }
3187     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3188 }
3189 \l_@@_notes_code_after_tl
3190 \seq_gclear:N \g_@@_notes_seq
3191 \seq_gclear:N \g_@@_notes_in_caption_seq
3192 \int_gzero:N \c@tabularnote
3193 }

```

The following command will format (after the main tabular) one tabularnote (with the command `\item`). `#1` is the label (when the command `\tabularnote` has been used with an optional argument between square brackets) and `#2` is the text of the note. The second argument is provided by curryfication.

```

3194 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3195 {
3196     \tl_if_novalue:nTF { #1 }
3197     { \item }
3198     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3199 }

```

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`.

```

3200 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3201 {
3202     \pgfpicture
3203     \@@_qpoint:n { row - 1 }
3204     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3205     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3206     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3207     \endpgfpicture
3208     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3209     \int_if_zero:nT { \l_@@_first_row_int }
3210     {
3211         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3212         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3213     }
3214     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3215 }

```

Now, the general case.

```

3216 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3217 {

```

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

```

3218     \str_if_eq:eeT \l_@@_baseline_tl { t }
3219     { \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`.

```

3220     \pgfpicture
3221     \@@_qpoint:n { row - 1 }
3222     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3223     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3224     {
3225         \int_set:Nn \l_tmpa_int
3226         {
3227             \str_range:Nnn
3228             \l_@@_baseline_tl
3229             { 6 }
3230             { \tl_count:o \l_@@_baseline_tl }

```

```

3231     }
3232     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3233   }
3234   {
3235     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3236     \bool_lazy_or:nnT
3237       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3238       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3239     {
3240       \@@_error:n { bad-value-for~baseline }
3241       \int_set:Nn \l_tmpa_int 1
3242     }
3243     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3244   }
3245   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3246   \endpgfpicture
3247   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3248   \int_if_zero:nT { \l_@@_first_row_int }
3249     {
3250       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3251       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3252     }
3253   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3254 }

```

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.

```

3255 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3256 {

```

We will compute the real width of both delimiters used.

```

3257   \dim_zero_new:N \l_@@_real_left_delim_dim
3258   \dim_zero_new:N \l_@@_real_right_delim_dim
3259   \hbox_set:Nn \l_tmpb_box
3260     {
3261     \m@th % added 2024/11/21
3262     \c_math_toggle_token
3263     \left #1
3264     \vcenter
3265       {
3266         \vbox_to_ht:nn
3267           { \box_ht_plus_dp:N \l_tmpa_box }
3268           { }
3269       }
3270     \right .
3271     \c_math_toggle_token
3272   }
3273   \dim_set:Nn \l_@@_real_left_delim_dim
3274     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3275   \hbox_set:Nn \l_tmpb_box
3276     {
3277     \m@th % added 2024/11/21
3278     \c_math_toggle_token
3279     \left .
3280     \vbox_to_ht:nn
3281       { \box_ht_plus_dp:N \l_tmpa_box }
3282       { }
3283     \right #2
3284     \c_math_toggle_token
3285   }
3286   \dim_set:Nn \l_@@_real_right_delim_dim
3287     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3288     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3289     \@@_put_box_in_flow:
3290     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3291 }

```

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).

```

3292 \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).

```

3293 {
3294   \peek_remove_spaces:n
3295   {
3296     \peek_meaning:NTF \end
3297     { \@@_analyze_end:Nn }
3298     {
3299       \@@_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`).

```

3300     \@@_array:o \g_@@_array_preamble_tl
3301   }
3302 }
3303 }
3304 {
3305   \@@_create_col_nodes:
3306   \endarray
3307 }

```

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

```

3308 \NewDocumentEnvironment { @@-light-syntax } { b }
3309 {

```

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`.

```

3310   \tl_if_empty:nT { #1 }
3311   { \@@_fatal:n { empty-environment } }
3312   \tl_if_in:nnT { #1 } { & }
3313   { \@@_fatal:n { ampersand-in-light-syntax } }
3314   \tl_if_in:nnT { #1 } { \ }
3315   { \@@_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`.

```

3316     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3317 }

```

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.

```

3318 {
3319   \@@_create_col_nodes:
3320   \endarray
3321 }
3322 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3323 {
3324   \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).

```

3325   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3326   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3327   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3328     { \seq_set_split:Nee }
3329     { \seq_set_split:Non }
3330   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3331   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3332   \tl_if_empty:NF \l_tmpa_tl
3333     { \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.

```

3334   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3335     { \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`).

```

3336   \tl_build_begin:N \l_@@_new_body_tl
3337   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3338   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3339   \@@_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).

```

3340   \seq_map_inline:Nn \l_@@_rows_seq
3341     {
3342       \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3343       \@@_line_with_light_syntax:n { ##1 }
3344     }
3345   \tl_build_end:N \l_@@_new_body_tl
3346   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3347     {
3348       \int_set:Nn \l_@@_last_col_int
3349         { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3350     }

```

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.

```

3351   \@@_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`).

```

3352   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3353 }

```

```

3354 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3355 {
3356   \seq_clear_new:N \l_@@_cells_seq
3357   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3358   \int_set:Nn \l_@@_nb_cols_int
3359   {
3360     \int_max:nn
3361     { \l_@@_nb_cols_int }
3362     { \seq_count:N \l_@@_cells_seq }
3363   }
3364   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3365   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3366   \seq_map_inline:Nn \l_@@_cells_seq
3367   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3368 }
3369 \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`.

```

3370 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3371 {
3372   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3373   { \@@_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.

```

3374   \end { #2 }
3375 }

```

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`).

```

3376 \cs_new:Npn \@@_create_col_nodes:
3377 {
3378   \crrc
3379   \int_if_zero:nT { \l_@@_first_col_int }
3380   {
3381     \omit
3382     \hbox_overlap_left:n
3383     {
3384       \bool_if:NT \l_@@_code_before_bool
3385       { \pgfsys@markposition { \@@_env: - col - 0 } }
3386       \pgfpicture
3387       \pgfrememberpicturepositiononpagetrue
3388       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3389       \str_if_empty:NF \l_@@_name_str
3390       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3391       \endpgfpicture
3392       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3393     }
3394     &
3395   }
3396   \omit

```

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

```

3397   \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`).

```

3398   \int_if_zero:nTF { \l_@@_first_col_int }
3399   {
3400     \bool_if:NT \l_@@_code_before_bool

```

```

3401     {
3402         \hbox
3403         {
3404             \skip_horizontal:n { -0.5 \arrayrulewidth }
3405             \pgfsys@markposition { \@@_env: - col - 1 }
3406             \skip_horizontal:n { 0.5 \arrayrulewidth }
3407         }
3408     }
3409     \pgfpicture
3410     \pgfrememberpicturepositiononpagetrue
3411     \pgfcoordinate { \@@_env: - col - 1 }
3412     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3413     \str_if_empty:NF \l_@@_name_str
3414     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3415     \endpgfpicture
3416 }
3417 {
3418     \bool_if:NT \l_@@_code_before_bool
3419     {
3420         \hbox
3421         {
3422             \skip_horizontal:n { 0.5 \arrayrulewidth }
3423             \pgfsys@markposition { \@@_env: - col - 1 }
3424             \skip_horizontal:n { -0.5 \arrayrulewidth }
3425         }
3426     }
3427     \pgfpicture
3428     \pgfrememberpicturepositiononpagetrue
3429     \pgfcoordinate { \@@_env: - col - 1 }
3430     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3431     \str_if_empty:NF \l_@@_name_str
3432     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3433     \endpgfpicture
3434 }

```

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.

```

3435     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3436     \bool_if:NF \l_@@_auto_columns_width_bool
3437     { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3438     {
3439         \bool_lazy_and:nnTF
3440         { \l_@@_auto_columns_width_bool }
3441         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3442         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3443         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3444         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3445     }
3446     \skip_horizontal:N \g_tmpa_skip
3447     \hbox
3448     {
3449         \bool_if:NT \l_@@_code_before_bool
3450         {
3451             \hbox
3452             {
3453                 \skip_horizontal:n { -0.5 \arrayrulewidth }
3454                 \pgfsys@markposition { \@@_env: - col - 2 }
3455                 \skip_horizontal:n { 0.5 \arrayrulewidth }
3456             }

```



```

3457     }
3458     \pgfpicture
3459     \pgfrememberpicturepositiononpagetrue
3460     \pgfcoordinate { \@@_env: - col - 2 }
3461     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3462     \str_if_empty:NF \l_@@_name_str
3463     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3464     \endpgfpicture
3465 }

```

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.

```

3466     \int_gset_eq:NN \g_tmpa_int \c_one_int
3467     \bool_if:NTF \g_@@_last_col_found_bool
3468     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3469     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3470     {
3471     &
3472     \omit
3473     \int_gincr:N \g_tmpa_int

```

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

```

3474     \skip_horizontal:N \g_tmpa_skip
3475     \bool_if:NT \l_@@_code_before_bool
3476     {
3477     \hbox
3478     {
3479     \skip_horizontal:n { -0.5 \arrayrulewidth }
3480     \pgfsys@markposition
3481     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3482     \skip_horizontal:n { 0.5 \arrayrulewidth }
3483     }
3484     }

```

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

```

3485     \pgfpicture
3486     \pgfrememberpicturepositiononpagetrue
3487     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3488     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3489     \str_if_empty:NF \l_@@_name_str
3490     {
3491     \pgfnodealias
3492     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3493     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3494     }
3495     \endpgfpicture
3496     }

3497     &
3498     \omit

```

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

```

3499     \int_if_zero:nT { \g_@@_col_total_int }
3500     { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3501     \skip_horizontal:N \g_tmpa_skip
3502     \int_gincr:N \g_tmpa_int
3503     \bool_lazy_any:nF
3504     {
3505     \g_@@_delims_bool
3506     \l_@@_tabular_bool
3507     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3508     \l_@@_exterior_arraycolsep_bool
3509     \l_@@_bar_at_end_of_pream_bool

```

```

3510     }
3511     { \skip_horizontal:n { - \col@sep } }
3512 \bool_if:NT \l_@@_code_before_bool
3513     {
3514         \hbox
3515         {
3516             \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.

```

3517         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3518         { \skip_horizontal:n { - \arraycolsep } }
3519         \pgfsys@markposition
3520         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3521         \skip_horizontal:n { 0.5 \arrayrulewidth }
3522         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3523         { \skip_horizontal:N \arraycolsep }
3524     }
3525 }
3526 \pgfpicture
3527 \pgfrememberpicturepositiononpagetrue
3528 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3529 {
3530     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3531     {
3532         \pgfpoint
3533         { - 0.5 \arrayrulewidth - \arraycolsep }
3534         \c_zero_dim
3535     }
3536     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3537 }
3538 \str_if_empty:NF \l_@@_name_str
3539 {
3540     \pgfnodealias
3541     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3542     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3543 }
3544 \endpgfpicture

3545 \bool_if:NT \g_@@_last_col_found_bool
3546 {
3547     \hbox_overlap_right:n
3548     {
3549         \skip_horizontal:N \g_@@_width_last_col_dim
3550         \skip_horizontal:N \col@sep
3551         \bool_if:NT \l_@@_code_before_bool
3552         {
3553             \pgfsys@markposition
3554             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3555         }
3556         \pgfpicture
3557         \pgfrememberpicturepositiononpagetrue
3558         \pgfcoordinate
3559         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3560         \pgfpointorigin
3561         \str_if_empty:NF \l_@@_name_str
3562         {
3563             \pgfnodealias
3564             {
3565                 \l_@@_name_str - col
3566                 - \int_eval:n { \g_@@_col_total_int + 1 }
3567             }

```

```

3568         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3569     }
3570     \endpgfpicture
3571 }
3572 }
3573 % \cr
3574 }

```

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

```

3575 \tl_const:Nn \c_@@_preamble_first_col_tl
3576 {
3577     >
3578     {

```

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

```

3579     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3580     \bool_gset_true:N \g_@@_after_col_zero_bool
3581     \@@_begin_of_row:
3582     \hbox_set:Nw \l_@@_cell_box
3583     \@@_math_toggle:
3584     \@@_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”.

```

3585     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3586     {
3587         \bool_lazy_or:nnT
3588         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3589         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3590         {
3591             \l_@@_code_for_first_col_tl
3592             \xglobal \colorlet { nicematrix-first-col } { . }
3593         }
3594     }
3595 }

```

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`.

```

3596     l
3597     <
3598     {
3599         \@@_math_toggle:
3600         \hbox_set_end:
3601         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3602         \@@_adjust_size_box:
3603         \@@_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.

```

3604     \dim_gset:Nn \g_@@_width_first_col_dim
3605     { \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.

```

3606     \hbox_overlap_left:n
3607     {
3608         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3609         { \@@_node_cell: }
3610         { \box_use_drop:N \l_@@_cell_box }
3611         \skip_horizontal:N \l_@@_left_delim_dim
3612         \skip_horizontal:N \l_@@_left_margin_dim
3613         \skip_horizontal:N \l_@@_extra_left_margin_dim
3614     }
3615     \bool_gset_false:N \g_@@_empty_cell_bool

```

```

3616     \skip_horizontal:n { -2 \col@sep }
3617   }
3618 }

```

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

```

3619 \tl_const:Nn \c_@@_preamble_last_col_tl
3620 {
3621   >
3622   {
3623     \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).

```

3624     \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.

```

3625     \bool_gset_true:N \g_@@_last_col_found_bool
3626     \int_gincr:N \c@jCol
3627     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3628     \hbox_set:Nw \l_@@_cell_box
3629     \@@_math_toggle:
3630     \@@_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”.

```

3631     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3632     {
3633       \bool_lazy_or:nnT
3634         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3635         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3636       {
3637         \l_@@_code_for_last_col_tl
3638         \xglobal \colorlet { nicematrix-last-col } { . }
3639       }
3640     }
3641   }
3642   l
3643   <
3644   {
3645     \@@_math_toggle:
3646     \hbox_set_end:
3647     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3648     \@@_adjust_size_box:
3649     \@@_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.

```

3650     \dim_gset:Nn \g_@@_width_last_col_dim
3651     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3652     \skip_horizontal:n { -2 \col@sep }

```

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

```

3653     \hbox_overlap_right:n
3654     {
3655       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3656       {
3657         \skip_horizontal:N \l_@@_right_delim_dim
3658         \skip_horizontal:N \l_@@_right_margin_dim
3659         \skip_horizontal:N \l_@@_extra_right_margin_dim
3660         \@@_node_cell:
3661       }
3662     }
3663     \bool_gset_false:N \g_@@_empty_cell_bool
3664   }
3665 }

```

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

```

3666 \NewDocumentEnvironment { NiceArray } { }
3667 {
3668   \bool_gset_false:N \g_@@_delims_bool
3669   \str_if_empty:NT \g_@@_name_env_str
3670   { \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).

```

3671   \NiceArrayWithDelims . .
3672 }
3673 { \endNiceArrayWithDelims }

```

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

```

3674 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3675 {
3676   \NewDocumentEnvironment { #1 NiceArray } { }
3677   {
3678     \bool_gset_true:N \g_@@_delims_bool
3679     \str_if_empty:NT \g_@@_name_env_str
3680     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3681     \@@_test_if_math_mode:
3682     \NiceArrayWithDelims #2 #3
3683   }
3684   { \endNiceArrayWithDelims }
3685 }
3686 \@@_def_env:NNN p ( )
3687 \@@_def_env:NNN b [ ]
3688 \@@_def_env:NNN B \{ \}
3689 \@@_def_env:NNN v | |
3690 \@@_def_env:NNN V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3691 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3692 {
3693   \bool_set_false:N \l_@@_preamble_bool
3694   \tl_clear:N \l_tmpa_tl
3695   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3696   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3697   \tl_put_right:Nn \l_tmpa_tl
3698   {
3699     *
3700     {
3701       \int_case:nnF \l_@@_last_col_int
3702       {
3703         { -2 } { \c@MaxMatrixCols }
3704         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3705       }
3706       { \int_eval:n { \l_@@_last_col_int - 1 } }
3707     }
3708     { #2 }
3709   }
3710   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3711   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3712 }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3713 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3714 \clist_map_inline:nn { p , b , B , v , V }
3715 {
3716   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3717   {
3718     \bool_gset_true:N \g_@@_delims_bool
3719     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3720     \int_if_zero:nT { \l_@@_last_col_int }
3721     {
3722       \bool_set_true:N \l_@@_last_col_without_value_bool
3723       \int_set:Nn \l_@@_last_col_int { -1 }
3724     }
3725     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3726     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3727   }
3728   { \use:c { end #1 NiceArray } }
3729 }

```

We define also an environment {NiceMatrix}

```

3730 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3731 {
3732   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3733   \int_if_zero:nT { \l_@@_last_col_int }
3734   {
3735     \bool_set_true:N \l_@@_last_col_without_value_bool
3736     \int_set:Nn \l_@@_last_col_int { -1 }
3737   }
3738   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3739   \bool_lazy_or:nnT
3740   { \clist_if_empty_p:N \l_@@_vlines_clist }
3741   { \l_@@_except_borders_bool }
3742   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3743   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3744 }
3745 { \endNiceArray }

```

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

```

3746 \cs_new_protected:Npn \@@_NotEmpty:
3747 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3748 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3749 {

```

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.

```

3750   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3751   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3752   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3753   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3754   \tl_if_empty:NF \l_@@_short_caption_tl
3755   {
3756     \tl_if_empty:NT \l_@@_caption_tl
3757     {
3758       \@@_error_or_warning:n { short-caption-without-caption }
3759       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3760     }
3761   }
3762   \tl_if_empty:NF \l_@@_label_tl
3763   {

```

```

3764     \tl_if_empty:NT \l_@@_caption_tl
3765     { \@@_error_or_warning:n { label~without~caption } }
3766   }
3767 \NewDocumentEnvironment { TabularNote } { b }
3768 {
3769   \bool_if:NTF \l_@@_in_code_after_bool
3770   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3771   {
3772     \tl_if_empty:NF \g_@@_tabularnote_tl
3773     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3774     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3775   }
3776 }
3777 { }
3778 \@@_settings_for_tabular:
3779 \NiceArray { #2 }
3780 }
3781 { \endNiceArray }
3782 \cs_new_protected:Npn \@@_settings_for_tabular:
3783 {
3784   \bool_set_true:N \l_@@_tabular_bool
3785   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3786   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3787   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3788 }

3789 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3790 {
3791   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3792   \dim_set:Nn \l_@@_width_dim { #1 }
3793   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3794   \@@_settings_for_tabular:
3795   \NiceArray { #3 }
3796 }
3797 {
3798   \endNiceArray
3799   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3800   { \@@_error:n { NiceTabularX~without~X } }
3801 }

3802 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3803 {
3804   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3805   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3806   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3807   \@@_settings_for_tabular:
3808   \NiceArray { #3 }
3809 }
3810 { \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`).

```

3811 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3812 {
3813   \bool_lazy_all:nT
3814   {

```

```

3815     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3816     { \l_@@_hvlines_bool }
3817     { ! \g_@@_delims_bool }
3818     { ! \l_@@_except_borders_bool }
3819   }
3820   {
3821     \bool_set_true:N \l_@@_except_borders_bool
3822     \clist_if_empty:NF \l_@@_corners_clist
3823     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3824     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3825     {
3826       \@@_stroke_block:nnn
3827       {
3828         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3829         draw = \l_@@_rules_color_tl
3830       }
3831       { 1-1 }
3832       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3833     }
3834   }
3835 }

```

```

3836 \cs_new_protected:Npn \@@_after_array:
3837 {

```

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`.

```

3838     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3839     \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.

```

3840     \bool_if:NT \g_@@_last_col_found_bool
3841     { \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`.

```

3842     \bool_if:NT \l_@@_last_col_without_value_bool
3843     { \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.

```

3844     \bool_if:NT \l_@@_last_row_without_value_bool
3845     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3846     \tl_gput_right:Ne \g_@@_aux_tl
3847     {
3848       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3849       {
3850         \int_use:N \l_@@_first_row_int ,
3851         \int_use:N \c@iRow ,
3852         \int_use:N \g_@@_row_total_int ,
3853         \int_use:N \l_@@_first_col_int ,
3854         \int_use:N \c@jCol ,
3855         \int_use:N \g_@@_col_total_int
3856       }
3857     }

```


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`).

```

3858   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3859   {
3860     \tl_gput_right:Ne \g_@@_aux_tl
3861     {
3862       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3863       { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3864     }
3865   }
3866 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3867 {
3868   \tl_gput_right:Ne \g_@@_aux_tl
3869   {
3870     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3871     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3872     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3873     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3874   }
3875 }

```

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

```

3876   \@@_create_diag_nodes:

```

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

```

3877   \pgfpicture
3878   \@@_create_aliases_last:
3879   \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3880   \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.

```

3881   \bool_if:NT \l_@@_parallelize_diags_bool
3882   {
3883     \int_gzero:N \g_@@_ddots_int
3884     \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.

```

3885     \dim_gzero:N \g_@@_delta_x_one_dim
3886     \dim_gzero:N \g_@@_delta_y_one_dim
3887     \dim_gzero:N \g_@@_delta_x_two_dim
3888     \dim_gzero:N \g_@@_delta_y_two_dim
3889   }
3890   \bool_set_false:N \l_@@_initial_open_bool
3891   \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.

```

3892   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3893   \@@_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.

```

3894   \clist_if_empty:NF \l_@@_corners_cells_clist
3895   {
3896     \bool_if:NTF \l_@@_no_cell_nodes_bool
3897     { \@@_error:n { corners~with~no~cell~nodes } }
3898     { \@@_compute_corners: }
3899   }

```

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

```

3900   \@@_adjust_pos_of_blocks_seq:
3901   \@@_deal_with_rounded_corners:
3902   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3903   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3904   \IfPackageLoadedT { tikz }
3905   {
3906     \tikzset
3907     {
3908       every-picture / .style =
3909       {
3910         overlay ,
3911         remember~picture ,
3912         name~prefix = \@@_env: -
3913       }
3914     }
3915   }
3916   \bool_if:NT \c_@@_recent_array_bool
3917   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3918   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3919   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3920   \cs_set_eq:NN \OverBrace \@@_OverBrace
3921   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3922   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3923   \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.

```

3924   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3925   \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.

```

3926   \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).

```

3927   \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`).

```

3928   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3929   { \@@_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:`.

```

3930   \bool_set_true:N \l_@@_in_code_after_bool
3931   \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3932   \scan_stop:
3933   \tl_gclear:N \g_nicematrix_code_after_tl
3934   \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.

```

3935   \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3936   \tl_if_empty:NF \g_@@_pre_code_before_tl
3937   {
3938     \tl_gput_right:Ne \g_@@_aux_tl
3939     {
3940       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3941       { \exp_not:o \g_@@_pre_code_before_tl }
3942     }
3943     \tl_gclear:N \g_@@_pre_code_before_tl
3944   }
3945   \tl_if_empty:NF \g_nicematrix_code_before_tl
3946   {
3947     \tl_gput_right:Ne \g_@@_aux_tl
3948     {
3949       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3950       { \exp_not:o \g_nicematrix_code_before_tl }
3951     }
3952     \tl_gclear:N \g_nicematrix_code_before_tl
3953   }

3954   \str_gclear:N \g_@@_name_env_str
3955   \@@_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`.

```

3956   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3957   }

3958   \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3959   {
3960     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3961     \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.

3962     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3963     { 0.6 \l_@@_xdots_shorten_start_dim }
3964     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3965     { 0.6 \l_@@_xdots_shorten_end_dim }
3966   }

```

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`.

```

3967   \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3968   { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

3969 \cs_new_protected:Npn \@@_create_alias_nodes:
3970 {
3971   \int_step_inline:nn { \c@iRow }
3972   {
3973     \pgfnodealias
3974     { \l_@@_name_str - ##1 - last }
3975     { \@@_env: - ##1 - \int_use:N \c@jCol }
3976   }
3977   \int_step_inline:nn { \c@jCol }
3978   {
3979     \pgfnodealias
3980     { \l_@@_name_str - last - ##1 }
3981     { \@@_env: - \int_use:N \c@iRow - ##1 }
3982   }
3983   \pgfnodealias % added 2025-04-05
3984   { \l_@@_name_str - last - last }
3985   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3986 }

```

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).

```

3987 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3988 {
3989   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3990   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3991 }

```

The following command must *not* be protected.

```

3992 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3993 {
3994   { #1 }
3995   { #2 }
3996   {
3997     \int_compare:nNnTF { #3 } > { 98 }
3998     { \int_use:N \c@iRow }
3999     { #3 }
4000   }
4001   {
4002     \int_compare:nNnTF { #4 } > { 98 }
4003     { \int_use:N \c@jCol }
4004     { #4 }
4005   }
4006   { #5 }
4007 }

```

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).

```

4008 \hook_gput_code:nnn { begindocument } { . }
4009 {
4010   \cs_new_protected:Npe \@@_draw_dotted_lines:
4011   {
4012     \c_@@_pgfortikzpicture_tl
4013     \@@_draw_dotted_lines_i:
4014     \c_@@_endpgfortikzpicture_tl
4015   }
4016 }

```

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

```

4017 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4018 {
4019   \pgfrememberpicturepositiononpagetrue
4020   \pgf@relevantforpicturesizefalse
4021   \g_@@_HVdotsfor_lines_tl
4022   \g_@@_Vdots_lines_tl
4023   \g_@@_Ddots_lines_tl
4024   \g_@@_Iddots_lines_tl
4025   \g_@@_Cdots_lines_tl
4026   \g_@@_Ldots_lines_tl
4027 }

4028 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4029 {
4030   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4031   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4032 }

```

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

```

4033 \pgfdeclareshape { @@_diag_node }
4034 {
4035   \savedanchor { \five }
4036   {
4037     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4038     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4039   }
4040   \anchor { 5 } { \five }
4041   \anchor { center } { \pgfpointorigin }
4042   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4043   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4044   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4045   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4046   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4047   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4048   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4049   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4050   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4051   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4052 }

```

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).

```

4053 \cs_new_protected:Npn \@@_create_diag_nodes:
4054 {
4055   \pgfpicture
4056   \pgfrememberpicturepositiononpagetrue
4057   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4058   {
4059     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4060     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4061     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4062     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4063     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4064     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4065     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4066     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4067     \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.

```

4068     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4069     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4070     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4071     \str_if_empty:NF \l_@@_name_str
4072     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4073 }

```

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

```

4074     \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4075     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4076     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4077     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4078     \pgfcoordinate
4079     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4080     \pgfnodealias
4081     { \@@_env: - last }
4082     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4083     \str_if_empty:NF \l_@@_name_str
4084     {
4085         \pgfnodealias
4086         { \l_@@_name_str - \int_use:N \l_tmpa_int }
4087         { \@@_env: - \int_use:N \l_tmpa_int }
4088         \pgfnodealias
4089         { \l_@@_name_str - last }
4090         { \@@_env: - last }
4091     }
4092     \endpgfpicture
4093 }

```

16 We draw the dotted lines

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

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

The command `\@@_find_extremities_of_line:nmnn` 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.

```
4094 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4095 {
```

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

```
4096 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4097 \int_set:Nn \l_@@_initial_i_int { #1 }
4098 \int_set:Nn \l_@@_initial_j_int { #2 }
4099 \int_set:Nn \l_@@_final_i_int { #1 }
4100 \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.

```
4101 \bool_set_false:N \l_@@_stop_loop_bool
4102 \bool_do_until:Nn \l_@@_stop_loop_bool
4103 {
4104 \int_add:Nn \l_@@_final_i_int { #3 }
4105 \int_add:Nn \l_@@_final_j_int { #4 }
4106 \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.

```
4107 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4108 \if_int_compare:w #3 = \c_one_int
4109 \bool_set_true:N \l_@@_final_open_bool
4110 \else:
4111 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4112 \bool_set_true:N \l_@@_final_open_bool
4113 \fi:
4114 \fi:
4115 \else:
4116 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4117 \if_int_compare:w #4 = -1
4118 \bool_set_true:N \l_@@_final_open_bool
4119 \fi:
4120 \else:
4121 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4122 \if_int_compare:w #4 = \c_one_int
4123 \bool_set_true:N \l_@@_final_open_bool
4124 \fi:
4125 \fi:
4126 \fi:
4127 \fi:
4128 \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.

```
4129 {
```

We do a step backwards.

```
4130 \int_sub:Nn \l_@@_final_i_int { #3 }
4131 \int_sub:Nn \l_@@_final_j_int { #4 }
4132 \bool_set_true:N \l_@@_stop_loop_bool
4133 }
```

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`.

```
4134 {
4135 \cs_if_exist:cTF
4136 {
4137 @@ _ dotted _
```

```

4138     \int_use:N \l_@@_final_i_int -
4139     \int_use:N \l_@@_final_j_int
4140   }
4141   {
4142     \int_sub:Nn \l_@@_final_i_int { #3 }
4143     \int_sub:Nn \l_@@_final_j_int { #4 }
4144     \bool_set_true:N \l_@@_final_open_bool
4145     \bool_set_true:N \l_@@_stop_loop_bool
4146   }
4147   {
4148     \cs_if_exist:cTF
4149     {
4150       pgf @ sh @ ns @ \@@_env:
4151       - \int_use:N \l_@@_final_i_int
4152       - \int_use:N \l_@@_final_j_int
4153     }
4154     { \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.

```

4155     {
4156       \cs_set_nopar:cpn
4157       {
4158         @@ _ dotted _
4159         \int_use:N \l_@@_final_i_int -
4160         \int_use:N \l_@@_final_j_int
4161       }
4162       { }
4163     }
4164   }
4165 }
4166 }

```

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

```

4167   \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4168   \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4169   \bool_do_until:Nn \l_@@_stop_loop_bool
4170   {
4171     \int_sub:Nn \l_@@_initial_i_int { #3 }
4172     \int_sub:Nn \l_@@_initial_j_int { #4 }
4173     \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.

```

4174     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4175     \if_int_compare:w #3 = \c_one_int
4176     \bool_set_true:N \l_@@_initial_open_bool
4177     \else:

```

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

```

4178     \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4179     \bool_set_true:N \l_@@_initial_open_bool
4180     \fi:
4181     \fi:
4182   \else:
4183     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4184     \if_int_compare:w #4 = \c_one_int

```



```

4185         \bool_set_true:N \l_@@_initial_open_bool
4186     \fi:
4187 \else:
4188     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4189         \if_int_compare:w #4 = -1
4190             \bool_set_true:N \l_@@_initial_open_bool
4191         \fi:
4192     \fi:
4193 \fi:
4194 \fi:
4195 \bool_if:NTF \l_@@_initial_open_bool
4196 {
4197     \int_add:Nn \l_@@_initial_i_int { #3 }
4198     \int_add:Nn \l_@@_initial_j_int { #4 }
4199     \bool_set_true:N \l_@@_stop_loop_bool
4200 }
4201 {
4202     \cs_if_exist:cTF
4203     {
4204         @@ _ dotted _
4205         \int_use:N \l_@@_initial_i_int -
4206         \int_use:N \l_@@_initial_j_int
4207     }
4208     {
4209         \int_add:Nn \l_@@_initial_i_int { #3 }
4210         \int_add:Nn \l_@@_initial_j_int { #4 }
4211         \bool_set_true:N \l_@@_initial_open_bool
4212         \bool_set_true:N \l_@@_stop_loop_bool
4213     }
4214     {
4215         \cs_if_exist:cTF
4216         {
4217             pgf @ sh @ ns @ \@@_env:
4218             - \int_use:N \l_@@_initial_i_int
4219             - \int_use:N \l_@@_initial_j_int
4220         }
4221         { \bool_set_true:N \l_@@_stop_loop_bool }
4222         {
4223             \cs_set_nopar:cpn
4224             {
4225                 @@ _ dotted _
4226                 \int_use:N \l_@@_initial_i_int -
4227                 \int_use:N \l_@@_initial_j_int
4228             }
4229             { }
4230         }
4231     }
4232 }
4233 }

```

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.

```

4234     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4235     {
4236         { \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`.

```

4237         { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4238         { \int_use:N \l_@@_final_i_int }
4239         { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4240         { } % for the name of the block
4241     }
4242 }

```

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.

```

4243 \cs_new_protected:Npn \@@_open_shorten:
4244 {
4245   \bool_if:NT \l_@@_initial_open_bool
4246     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4247   \bool_if:NT \l_@@_final_open_bool
4248     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4249 }

```

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).

```

4250 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4251 {
4252   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4253   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4254   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4255   \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`.

```

4256   \seq_if_empty:NF \g_@@_submatrix_seq
4257   {
4258     \seq_map_inline:Nn \g_@@_submatrix_seq
4259       { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4260   }
4261 }

```

`#1` and `#2` are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. `#3`, `#4`, `#5` and `#6` are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #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.

```

4262 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4263 {
4264   \if_int_compare:w #3 > #1
4265   \else:
4266     \if_int_compare:w #1 > #5

```

```

4267 \else:
4268 \if_int_compare:w #4 > #2
4269 \else:
4270 \if_int_compare:w #2 > #6
4271 \else:
4272 \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4273 \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4274 \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4275 \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4276 \fi:
4277 \fi:
4278 \fi:
4279 \fi:
4280 }

4281 \cs_new_protected:Npn \@@_set_initial_coords:
4282 {
4283 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4284 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4285 }
4286 \cs_new_protected:Npn \@@_set_final_coords:
4287 {
4288 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4289 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4290 }
4291 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4292 {
4293 \pgfpointanchor
4294 {
4295 \@@_env:
4296 - \int_use:N \l_@@_initial_i_int
4297 - \int_use:N \l_@@_initial_j_int
4298 }
4299 { #1 }
4300 \@@_set_initial_coords:
4301 }
4302 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4303 {
4304 \pgfpointanchor
4305 {
4306 \@@_env:
4307 - \int_use:N \l_@@_final_i_int
4308 - \int_use:N \l_@@_final_j_int
4309 }
4310 { #1 }
4311 \@@_set_final_coords:
4312 }
4313 \cs_new_protected:Npn \@@_open_x_initial_dim:
4314 {
4315 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4316 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4317 {
4318 \cs_if_exist:cT
4319 { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4320 {
4321 \pgfpointanchor
4322 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4323 { west }
4324 \dim_set:Nn \l_@@_x_initial_dim
4325 { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4326 }
4327 }

```

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

```

4328 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4329 {
4330   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4331   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4332   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4333 }
4334 }
4335 \cs_new_protected:Npn \@@_open_x_final_dim:
4336 {
4337   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4338   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4339   {
4340     \cs_if_exist:cT
4341     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4342     {
4343       \pgfpointanchor
4344       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4345       { east }
4346       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4347       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4348     }
4349   }

```

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

```

4350 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4351 {
4352   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4353   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4354   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4355 }
4356 }

```

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.

```

4357 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4358 {
4359   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4360   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4361   {
4362     \@@_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.

```

4363 \group_begin:
4364 \@@_open_shorten:
4365 \int_if_zero:nTF { #1 }
4366 { \color { nicematrix-first-row } }
4367 {

```

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.

```

4368 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4369 { \color { nicematrix-last-row } }
4370 }
4371 \keys_set:nn { nicematrix / xdots } { #3 }
4372 \@@_color:o \l_@@_xdots_color_tl
4373 \@@_actually_draw_Ldots:
4374 \group_end:
4375 }
4376 }

```

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`.

```

4377 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4378 {
4379   \bool_if:NTF \l_@@_initial_open_bool
4380   {
4381     \@@_open_x_initial_dim:
4382     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4383     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4384   }
4385   { \@@_set_initial_coords_from_anchor:n { base-east } }
4386 \bool_if:NTF \l_@@_final_open_bool
4387 {
4388   \@@_open_x_final_dim:
4389   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4390   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4391 }
4392 { \@@_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.

```

4393 \bool_lazy_all:nTF
4394 {
4395   \l_@@_initial_open_bool
4396   \l_@@_final_open_bool
4397   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4398 }
4399 {
4400   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4401   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4402 }

```

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 (?).

```

4403 {
4404   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4405   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4406 }
4407 \@@_draw_line:
4408 }

```

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.

```

4409 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4410 {
4411   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4412   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4413   {
4414     \@@_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.

```

4415     \group_begin:
4416     \@@_open_shorten:
4417     \int_if_zero:nTF { #1 }
4418     { \color { nicematrix-first-row } }
4419     {

```

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.

```

4420         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4421         { \color { nicematrix-last-row } }
4422     }
4423     \keys_set:nn { nicematrix / xdots } { #3 }
4424     \@@_color:o \l_@@_xdots_color_tl
4425     \@@_actually_draw_Cdots:
4426 \group_end:
4427 }
4428 }

```

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`.

```

4429 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4430 {
4431     \bool_if:NTF \l_@@_initial_open_bool
4432     { \@@_open_x_initial_dim: }
4433     { \@@_set_initial_coords_from_anchor:n { mid-east } } }
4434 \bool_if:NTF \l_@@_final_open_bool
4435     { \@@_open_x_final_dim: }
4436     { \@@_set_final_coords_from_anchor:n { mid-west } } }
4437 \bool_lazy_and:mnTF
4438 { \l_@@_initial_open_bool }
4439 { \l_@@_final_open_bool }
4440 {
4441     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4442     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4443     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } } }
4444 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4445 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4446 }
4447 {
4448     \bool_if:NT \l_@@_initial_open_bool
4449     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4450     \bool_if:NT \l_@@_final_open_bool
4451     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4452 }
4453 \@@_draw_line:
4454 }
4455 \cs_new_protected:Npn \@@_open_y_initial_dim:
4456 {
4457     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4458     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4459     {

```

```

4460 \cs_if_exist:cT
4461 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4462 {
4463 \pgfpointanchor
4464 { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4465 { north }
4466 \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4467 { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4468 }
4469 }
4470 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4471 {
4472 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4473 \dim_set:Nn \l_@@_y_initial_dim
4474 {
4475 \fp_to_dim:n
4476 {
4477 \pgf@y
4478 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4479 }
4480 }
4481 }
4482 }
4483 \cs_new_protected:Npn \@@_open_y_final_dim:
4484 {
4485 \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4486 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4487 {
4488 \cs_if_exist:cT
4489 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4490 {
4491 \pgfpointanchor
4492 { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4493 { south }
4494 \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4495 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4496 }
4497 }
4498 \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4499 {
4500 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4501 \dim_set:Nn \l_@@_y_final_dim
4502 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4503 }
4504 }

```

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.

```

4505 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4506 {
4507 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4508 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4509 {
4510 \@@_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.

```

4511 \group_begin:
4512 \@@_open_shorten:
4513 \int_if_zero:nTF { #2 }
4514 { \color { nicematrix-first-col } }
4515 {
4516 \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4517 { \color { nicematrix-last-col } }

```

```

4518     }
4519     \keys_set:nn { nicematrix / xdots } { #3 }
4520     \@@_color:o \l_@@_xdots_color_tl
4521     \@@_actually_draw_Vdots:
4522     \group_end:
4523   }
4524 }

```

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`.

```

4525 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4526 {

```

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

```

4527   \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.

```

4528   {
4529     \@@_open_y_initial_dim:
4530     \@@_open_y_final_dim:
4531     \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4532     {
4533       \@@_qpoint:n { col - 1 }
4534       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4535       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4536       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4537       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4538     }
4539     {
4540       \bool_lazy_and:nnTF
4541       { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4542       {
4543         \int_compare_p:nNn
4544         { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4545       }

```

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

```

4546     {
4547       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4548       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4549       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4550       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4551       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4552     }

```

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

```

4553     {
4554       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4555       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4556       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4557       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }

```



```

4558     }
4559   }
4560 }

```

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.

```

4561 {
4562   \bool_set_false:N \l_tmpa_bool
4563   \bool_if:NF \l_@@_initial_open_bool
4564   {
4565     \bool_if:NF \l_@@_final_open_bool
4566     {
4567       \@@_set_initial_coords_from_anchor:n { south-west }
4568       \@@_set_final_coords_from_anchor:n { north-west }
4569       \bool_set:Nn \l_tmpa_bool
4570       {
4571         \dim_compare_p:nNn
4572         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4573       }
4574     }
4575   }

```

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

```

4576   \bool_if:NTF \l_@@_initial_open_bool
4577   {
4578     \@@_open_y_initial_dim:
4579     \@@_set_final_coords_from_anchor:n { north }
4580     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4581   }
4582   {
4583     \@@_set_initial_coords_from_anchor:n { south }
4584     \bool_if:NTF \l_@@_final_open_bool
4585     { \@@_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.

```

4586   {
4587     \@@_set_final_coords_from_anchor:n { north }
4588     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4589     {
4590       \dim_set:Nn \l_@@_x_initial_dim
4591       {
4592         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4593         \l_@@_x_initial_dim \l_@@_x_final_dim
4594       }
4595     }
4596   }
4597 }
4598 }
4599 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4600 \@@_draw_line:
4601 }

```

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.

```

4602 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4603 {
4604   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4605   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4606   {
4607     \@@_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.

```

4608     \group_begin:
4609     \@@_open_shorten:
4610     \keys_set:nm { nicematrix / xdots } { #3 }
4611     \@@_color:o \l_@@_xdots_color_tl
4612     \@@_actually_draw_Ddots:
4613     \group_end:
4614   }
4615 }

```

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.`

```

4616 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4617 {
4618   \bool_if:NTF \l_@@_initial_open_bool
4619   {
4620     \@@_open_y_initial_dim:
4621     \@@_open_x_initial_dim:
4622   }
4623   { \@@_set_initial_coords_from_anchor:n { south-east } }
4624   \bool_if:NTF \l_@@_final_open_bool
4625   {
4626     \@@_open_x_final_dim:
4627     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4628   }
4629   { \@@_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.

```

4630   \bool_if:NT \l_@@_parallelize_diags_bool
4631   {
4632     \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).

```

4633     \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.

```

4634     {
4635       \dim_gset:Nn \g_@@_delta_x_one_dim
4636       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4637       \dim_gset:Nn \g_@@_delta_y_one_dim
4638       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4639     }

```

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`.

```

4640     {
4641       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4642       {
4643         \dim_set:Nn \l_@@_y_final_dim

```

```

4644         {
4645             \l_@@_y_initial_dim +
4646             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4647             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4648         }
4649     }
4650 }
4651 }
4652 \@@_draw_line:
4653 }

```

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.

```

4654 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4655 {
4656     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4657     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4658     {
4659         \@@_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.

```

4660         \group_begin:
4661         \@@_open_shorten:
4662         \keys_set:nn { nicematrix / xdots } { #3 }
4663         \@@_color:o \l_@@_xdots_color_tl
4664         \@@_actually_draw_Iddots:
4665     \group_end:
4666 }
4667 }

```

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`.

```

4668 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4669 {
4670     \bool_if:NTF \l_@@_initial_open_bool
4671     {
4672         \@@_open_y_initial_dim:
4673         \@@_open_x_initial_dim:
4674     }
4675     { \@@_set_initial_coords_from_anchor:n { south-west } }
4676     \bool_if:NTF \l_@@_final_open_bool
4677     {
4678         \@@_open_y_final_dim:
4679         \@@_open_x_final_dim:
4680     }
4681     { \@@_set_final_coords_from_anchor:n { north-east } }
4682     \bool_if:NT \l_@@_parallelize_diags_bool
4683     {
4684         \int_gincr:N \g_@@_iddots_int
4685         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }

```

```

4686     {
4687     \dim_gset:Nn \g_@@_delta_x_two_dim
4688     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4689     \dim_gset:Nn \g_@@_delta_y_two_dim
4690     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4691     }
4692     {
4693     \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4694     {
4695     \dim_set:Nn \l_@@_y_final_dim
4696     {
4697     \l_@@_y_initial_dim +
4698     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4699     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4700     }
4701     }
4702     }
4703     }
4704     \@@_draw_line:
4705     }

```

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`

```

4706 \cs_new_protected:Npn \@@_draw_line:
4707 {
4708 \pgfrememberpicturepositiononpagetrue
4709 \pgf@relevantforpicturesizefalse
4710 \bool_lazy_or:nnTF
4711 { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4712 { \l_@@_dotted_bool }
4713 { \@@_draw_standard_dotted_line: }
4714 { \@@_draw_unstandard_dotted_line: }
4715 }

```

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.

```

4716 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4717 {
4718 \begin { scope }
4719 \@@_draw_unstandard_dotted_line:o
4720 { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4721 }

```

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.

```

4722 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4723   {
4724     \@@_draw_unstandard_dotted_line:nooo
4725     { #1 }
4726     \l_@@_xdots_up_tl
4727     \l_@@_xdots_down_tl
4728     \l_@@_xdots_middle_tl
4729   }
4730 \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.

```

4731 \hook_gput_code:nnn { begindocument } { . }
4732 {
4733   \IfPackageLoadedT { tikz }
4734   {
4735     \tikzset
4736     {
4737       @@_node_above / .style = { sloped , above } ,
4738       @@_node_below / .style = { sloped , below } ,
4739       @@_node_middle / .style =
4740       {
4741         sloped ,
4742         inner~sep = \c_@@_innersep_middle_dim
4743       }
4744     }
4745   }
4746 }

4747 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4748   {

```

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.

```

4749   \dim_zero_new:N \l_@@_l_dim
4750   \dim_set:Nn \l_@@_l_dim
4751   {
4752     \fp_to_dim:n
4753     {
4754       sqrt
4755       (
4756         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4757         +
4758         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4759       )
4760     }
4761   }

```

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.

```

4762   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4763   {
4764     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }

```

```

4765     \l_@@_draw_unstandard_dotted_line_i:
4766 }

```

If the key `xdots/horizontal-labels` has been used.

```

4767 \bool_if:NT \l_@@_xdots_h_labels_bool
4768 {
4769   \tikzset
4770   {
4771     @@_node_above / .style = { auto = left } ,
4772     @@_node_below / .style = { auto = right } ,
4773     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4774   }
4775 }
4776 \tl_if_empty:nF { #4 }
4777 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4778 \draw
4779 [ #1 ]
4780 ( \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).

```

4781   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4782   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4783   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4784   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4785 \end { scope }
4786 }
4787 \cs_generate_variant:Nn \l_@@_draw_unstandard_dotted_line:nnnn { n o o o }
4788 \cs_new_protected:Npn \l_@@_draw_unstandard_dotted_line_i:
4789 {
4790   \dim_set:Nn \l_tmpa_dim
4791   {
4792     \l_@@_x_initial_dim
4793     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4794     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4795   }
4796   \dim_set:Nn \l_tmpb_dim
4797   {
4798     \l_@@_y_initial_dim
4799     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4800     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4801   }
4802   \dim_set:Nn \l_@@_tmpc_dim
4803   {
4804     \l_@@_x_final_dim
4805     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4806     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4807   }
4808   \dim_set:Nn \l_@@_tmpd_dim
4809   {
4810     \l_@@_y_final_dim
4811     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4812     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4813   }
4814   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4815   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4816   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4817   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4818 }

```

The command `\l_@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4819 \cs_new_protected:Npn \l_@@_draw_standard_dotted_line:

```

```

4820 {
4821   \group_begin:

```

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.

```

4822   \dim_zero_new:N \l_@@_l_dim
4823   \dim_set:Nn \l_@@_l_dim
4824     {
4825     \fp_to_dim:n
4826       {
4827         sqrt
4828         (
4829           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4830           +
4831           ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4832         )
4833       }
4834     }

```

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.

```

4835   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4836     {
4837       \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4838       { \@@_draw_standard_dotted_line_i: }
4839     }
4840   \group_end:
4841   \bool_lazy_all:nF
4842     {
4843     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4844     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4845     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4846     }
4847     { \@@_labels_standard_dotted_line: }
4848   }
4849   \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4850   \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4851     {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4852     \int_set:Nn \l_tmpa_int
4853       {
4854         \dim_ratio:nn
4855           {
4856             \l_@@_l_dim
4857             - \l_@@_xdots_shorten_start_dim
4858             - \l_@@_xdots_shorten_end_dim
4859           }
4860           { \l_@@_xdots_inter_dim }
4861       }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4862     \dim_set:Nn \l_tmpa_dim
4863       {
4864         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4865         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4866       }
4867     \dim_set:Nn \l_tmpb_dim
4868       {
4869         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

4870     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4871 }

```

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.

```

4872 \dim_gadd:Nn \l_@@_x_initial_dim
4873 {
4874   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4875   \dim_ratio:nn
4876   {
4877     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4878     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4879   }
4880   { 2 \l_@@_l_dim }
4881 }
4882 \dim_gadd:Nn \l_@@_y_initial_dim
4883 {
4884   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4885   \dim_ratio:nn
4886   {
4887     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4888     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4889   }
4890   { 2 \l_@@_l_dim }
4891 }
4892 \pgf@relevantforpicturesizefalse
4893 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4894 {
4895   \pgfpathcircle
4896   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4897   { \l_@@_xdots_radius_dim }
4898   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4899   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4900 }
4901 \pgfusepathqfill
4902 }

```

```

4903 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4904 {
4905   \pgfscope
4906   \pgftransformshift
4907   {
4908     \pgfpointlineattime { 0.5 }
4909     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4910     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4911   }
4912   \fp_set:Nn \l_tmpa_fp
4913   {
4914     atand
4915     (
4916       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4917       \l_@@_x_final_dim - \l_@@_x_initial_dim
4918     )
4919   }
4920   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4921   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4922   \tl_if_empty:NF \l_@@_xdots_middle_tl
4923   {
4924     \begin { pgfscope }
4925     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4926     \pgfnode
4927     { rectangle }
4928     { center }

```



```

4929     {
4930     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4931     {
4932     \c_math_toggle_token
4933     \scriptstyle \l_@@_xdots_middle_tl
4934     \c_math_toggle_token
4935     }
4936     }
4937     { }
4938     {
4939     \pgfsetfillcolor { white }
4940     \pgfusepath { fill }
4941     }
4942     \end { pgfscope }
4943     }
4944     \tl_if_empty:NF \l_@@_xdots_up_tl
4945     {
4946     \pgfnode
4947     { rectangle }
4948     { south }
4949     {
4950     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4951     {
4952     \c_math_toggle_token
4953     \scriptstyle \l_@@_xdots_up_tl
4954     \c_math_toggle_token
4955     }
4956     }
4957     { }
4958     { \pgfusepath { } }
4959     }
4960     \tl_if_empty:NF \l_@@_xdots_down_tl
4961     {
4962     \pgfnode
4963     { rectangle }
4964     { north }
4965     {
4966     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4967     {
4968     \c_math_toggle_token
4969     \scriptstyle \l_@@_xdots_down_tl
4970     \c_math_toggle_token
4971     }
4972     }
4973     { }
4974     { \pgfusepath { } }
4975     }
4976     \endpgfscope
4977     }

```

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.

```
4978 \hook_gput_code:nnn { begindocument } { . }
4979 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
4980 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4981 \cs_new_protected:Npn \@@_Ldots:
4982 { \@@_collect_options:n { \@@_Ldots_i } }
4983 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4984 {
4985 \int_if_zero:nTF { \c@jCol }
4986 { \@@_error:nn { in~first~col } { \Ldots } }
4987 {
4988 \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
4989 { \@@_error:nn { in~last~col } { \Ldots } }
4990 {
4991 \@@_instruction_of_type:nnn { \c_false_bool } { Ldots }
4992 { #1 , down = #2 , up = #3 , middle = #4 }
4993 }
4994 }
4995 \bool_if:NF \l_@@_nullify_dots_bool
4996 { \phantom { \ensuremath { \@@_old_ldots: } } }
4997 \bool_gset_true:N \g_@@_empty_cell_bool
4998 }
```

```
4999 \cs_new_protected:Npn \@@_Cdots:
5000 { \@@_collect_options:n { \@@_Cdots_i } }
5001 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5002 {
5003 \int_if_zero:nTF { \c@jCol }
5004 { \@@_error:nn { in~first~col } { \Cdots } }
5005 {
5006 \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5007 { \@@_error:nn { in~last~col } { \Cdots } }
5008 {
5009 \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5010 { #1 , down = #2 , up = #3 , middle = #4 }
5011 }
5012 }
5013 \bool_if:NF \l_@@_nullify_dots_bool
5014 { \phantom { \ensuremath { \@@_old_cdots: } } }
5015 \bool_gset_true:N \g_@@_empty_cell_bool
5016 }
```

```
5017 \cs_new_protected:Npn \@@_Vdots:
5018 { \@@_collect_options:n { \@@_Vdots_i } }
5019 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5020 {
5021 \int_if_zero:nTF { \c@iRow }
5022 { \@@_error:nn { in~first~row } { \Vdots } }
5023 {
5024 \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5025 { \@@_error:nn { in~last~row } { \Vdots } }
5026 {
5027 \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5028 { #1 , down = #2 , up = #3 , middle = #4 }
5029 }
5030 }
5031 \bool_if:NF \l_@@_nullify_dots_bool
5032 { \phantom { \ensuremath { \@@_old_vdots: } } } }
```

```

5033     \bool_gset_true:N \g_@@_empty_cell_bool
5034 }

5035 \cs_new_protected:Npn \@@_Ddots:
5036 { \@@_collect_options:n { \@@_Ddots_i } }
5037 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5038 {
5039     \int_case:nnF \c@iRow
5040     {
5041         0 { \@@_error:nn { in-first~row } { \Ddots } }
5042         \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Ddots } }
5043     }
5044     {
5045         \int_case:nnF \c@jCol
5046         {
5047             0 { \@@_error:nn { in-first~col } { \Ddots } }
5048             \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Ddots } }
5049         }
5050         {
5051             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5052             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5053             { #1 , down = #2 , up = #3 , middle = #4 }
5054         }
5055     }
5056 }
5057 \bool_if:NF \l_@@_nullify_dots_bool
5058 { \phantom { \ensuremath { \@@_old_ddots: } } }
5059 \bool_gset_true:N \g_@@_empty_cell_bool
5060 }

5061 \cs_new_protected:Npn \@@_Iddots:
5062 { \@@_collect_options:n { \@@_Iddots_i } }
5063 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5064 {
5065     \int_case:nnF \c@iRow
5066     {
5067         0 { \@@_error:nn { in-first~row } { \Iddots } }
5068         \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Iddots } }
5069     }
5070     {
5071         \int_case:nnF \c@jCol
5072         {
5073             0 { \@@_error:nn { in-first~col } { \Iddots } }
5074             \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Iddots } }
5075         }
5076         {
5077             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5078             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5079             { #1 , down = #2 , up = #3 , middle = #4 }
5080         }
5081     }
5082     \bool_if:NF \l_@@_nullify_dots_bool
5083     { \phantom { \ensuremath { \@@_old_iddots: } } }
5084     \bool_gset_true:N \g_@@_empty_cell_bool
5085 }
5086 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5087 \keys_define:nn { nicematrix / Ddots }
5088 {

```

```

5089     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5090     draw-first .default:n = true ,
5091     draw-first .value_forbidden:n = true
5092 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5093 \cs_new_protected:Npn \@@_Hspace:
5094 {
5095   \bool_gset_true:N \g_@@_empty_cell_bool
5096   \hspace
5097 }

```

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`.

```

5098 \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`.

```

5099 \cs_new:Npn \@@_Hdotsfor:
5100 {
5101   \bool_lazy_and:nnTF
5102     { \int_if_zero_p:n { \c@jCol } }
5103     { \int_if_zero_p:n { \l_@@_first_col_int } }
5104     {
5105       \bool_if:NTF \g_@@_after_col_zero_bool
5106       {
5107         \multicolumn { 1 } { c } { }
5108         \@@_Hdotsfor_i:
5109       }
5110       { \@@_fatal:n { Hdotsfor~in~col-0 } }
5111     }
5112   {
5113     \multicolumn { 1 } { c } { }
5114     \@@_Hdotsfor_i:
5115   }
5116 }

```

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:`).

```

5117 \hook_gput_code:nnn { begindocument } { . }
5118 {

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5119   \cs_new_protected:Npn \@@_Hdotsfor_i:
5120     { \@@_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`).

```

5121   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5122   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5123     {
5124     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5125     {
5126       \@@_Hdotsfor:nnnn
5127       { \int_use:N \c@iRow }
5128       { \int_use:N \c@jCol }
5129       { #2 }
5130       {
5131         #1 , #3 ,

```

```

5132         down = \exp_not:n { #4 } ,
5133         up = \exp_not:n { #5 } ,
5134         middle = \exp_not:n { #6 }
5135     }
5136 }
5137 \prg_replicate:nn { #2 - 1 }
5138 {
5139     &
5140     \multicolumn { 1 } { c } { }
5141     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5142 }
5143 }
5144 }

```

```

5145 \cs_new_protected:Npn \@@_Hdotsfor:nmmn #1 #2 #3 #4
5146 {
5147     \bool_set_false:N \l_@@_initial_open_bool
5148     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5149     \int_set:Nn \l_@@_initial_i_int { #1 }
5150     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5151     \int_compare:nNnTF { #2 } = { \c_one_int }
5152     {
5153         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5154         \bool_set_true:N \l_@@_initial_open_bool
5155     }
5156     {
5157         \cs_if_exist:cTF
5158         {
5159             pgf @ sh @ ns @ \@@_env:
5160             - \int_use:N \l_@@_initial_i_int
5161             - \int_eval:n { #2 - 1 }
5162         }
5163         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5164         {
5165             \int_set:Nn \l_@@_initial_j_int { #2 }
5166             \bool_set_true:N \l_@@_initial_open_bool
5167         }
5168     }
5169     \int_compare:nNnTF { #2 + #3 - 1 } = { \c_jCol }
5170     {
5171         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5172         \bool_set_true:N \l_@@_final_open_bool
5173     }
5174     {
5175         \cs_if_exist:cTF
5176         {
5177             pgf @ sh @ ns @ \@@_env:
5178             - \int_use:N \l_@@_final_i_int
5179             - \int_eval:n { #2 + #3 }
5180         }
5181         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5182         {
5183             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5184             \bool_set_true:N \l_@@_final_open_bool
5185         }
5186     }
5187     \group_begin:
5188     \@@_open_shorten:
5189     \int_if_zero:nTF { #1 }
5190     { \color { nicematrix-first-row } }

```

```

5191     {
5192       \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5193       { \color { nicematrix-last-row } }
5194     }
5195     \keys_set:nn { nicematrix / xdots } { #4 }
5196     \@@_color:o \l_@@_xdots_color_tl
5197     \@@_actually_draw_Ldots:
5198     \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).

```

5199     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5200     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5201   }

```

```

5202 \hook_gput_code:nnn { begindocument } { . }
5203 {
5204   \cs_new_protected:Npn \@@_Vdotsfor:
5205     { \@@_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`).

```

5206   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5207   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5208   {
5209     \bool_gset_true:N \g_@@_empty_cell_bool
5210     \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5211     {
5212       \@@_Vdotsfor:nnnn
5213       { \int_use:N \c@iRow }
5214       { \int_use:N \c@jCol }
5215       { #2 }
5216       {
5217         #1 , #3 ,
5218         down = \exp_not:n { #4 } ,
5219         up = \exp_not:n { #5 } ,
5220         middle = \exp_not:n { #6 }
5221       }
5222     }
5223   }
5224 }

```

```

5225 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5226 {
5227   \bool_set_false:N \l_@@_initial_open_bool
5228   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5229   \int_set:Nn \l_@@_initial_j_int { #2 }
5230   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

5231   \int_compare:nNnTF { #1 } = { \c_one_int }
5232   {
5233     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5234     \bool_set_true:N \l_@@_initial_open_bool
5235   }
5236   {
5237     \cs_if_exist:cTF
5238     {
5239       pgf @ sh @ ns @ \@@_env:

```

```

5240         - \int_eval:n { #1 - 1 }
5241         - \int_use:N \l_@@_initial_j_int
5242     }
5243     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5244     {
5245         \int_set:Nn \l_@@_initial_i_int { #1 }
5246         \bool_set_true:N \l_@@_initial_open_bool
5247     }
5248 }
5249 \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5250 {
5251     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5252     \bool_set_true:N \l_@@_final_open_bool
5253 }
5254 {
5255     \cs_if_exist:cTF
5256     {
5257         pgf @ sh @ ns @ \@@_env:
5258         - \int_eval:n { #1 + #3 }
5259         - \int_use:N \l_@@_final_j_int
5260     }
5261     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5262     {
5263         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5264         \bool_set_true:N \l_@@_final_open_bool
5265     }
5266 }
5267 \group_begin:
5268 \@@_open_shorten:
5269 \int_if_zero:nTF { #2 }
5270 { \color { nicematrix-first-col } }
5271 {
5272     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5273     { \color { nicematrix-last-col } }
5274 }
5275 \keys_set:nn { nicematrix / xdots } { #4 }
5276 \@@_color:o \l_@@_xdots_color_tl
5277 \@@_actually_draw_Vdots:
5278 \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).

```

5279     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5280     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5281 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5282 \NewDocumentCommand \@@_rotate: { 0 { } }
5283 {
5284     \bool_gset_true:N \g_@@_rotate_bool
5285     \keys_set:nn { nicematrix / rotate } { #1 }
5286     \ignorespaces
5287 }
5288 \keys_define:nn { nicematrix / rotate }
5289 {
5290     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5291     c .value_forbidden:n = true ,
5292     unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5293 }

```

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

```

5294 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5295   {
5296     \tl_if_empty:nTF { #2 }
5297       { #1 }
5298       { \@@_double_int_eval_i:n #1-#2 \q_stop }
5299   }
5300 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5301   { \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).

```

5302 \hook_gput_code:nnn { begindocument } { . }
5303   {

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5304   \tl_set_rescan:Nnn \l_tmpa_tl { }
5305   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5306   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5307     {
5308       \group_begin:
5309       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5310       \@@_color:o \l_@@_xdots_color_tl
5311       \use:e
5312         {
5313           \@@_line_i:nn
5314             { \@@_double_int_eval:n #2 - \q_stop }
5315             { \@@_double_int_eval:n #3 - \q_stop }
5316         }
5317       \group_end:
5318     }
5319 }

5320 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5321   {
5322     \bool_set_false:N \l_@@_initial_open_bool
5323     \bool_set_false:N \l_@@_final_open_bool
5324     \bool_lazy_or:nnTF
5325       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5326       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5327       { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5328     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5329   }

```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.


```

5330 \hook_gput_code:nnn { begindocument } { . }
5331 {
5332   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5333   {

```

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:`.

```

5334     \c_@@_pgfortikzpicture_tl
5335     \@@_draw_line_iii:nn { #1 } { #2 }
5336     \c_@@_endpgfortikzpicture_tl
5337   }
5338 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5339 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5340 {
5341   \pgfrememberpicturepositiononpagetrue
5342   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5343   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5344   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5345   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5346   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5347   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5348   \@@_draw_line:
5349 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```

5350 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5351 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```

5352 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5353 {
5354   \tl_gput_right:Ne \g_@@_row_style_tl
5355   {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can’t be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5356     \exp_not:N
5357     \@@_if_row_less_than:nn
5358     { \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`).

```

5359     {
5360         \exp_not:N
5361         \@@_if_col_greater_than:nn
5362         { \int_eval:n { \c@jCol } }
5363         { \exp_not:n { #1 } \scan_stop: }
5364     }
5365 }
5366 }
5367 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5368 \keys_define:nn { nicematrix / RowStyle }
5369 {
5370     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5371     cell-space-top-limit .value_required:n = true ,
5372     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5373     cell-space-bottom-limit .value_required:n = true ,
5374     cell-space-limits .meta:n =
5375     {
5376         cell-space-top-limit = #1 ,
5377         cell-space-bottom-limit = #1 ,
5378     } ,
5379     color .tl_set:N = \l_@@_color_tl ,
5380     color .value_required:n = true ,
5381     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5382     bold .default:n = true ,
5383     nb-rows .code:n =
5384     \str_if_eq:eeTF { #1 } { * }
5385     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5386     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5387     nb-rows .value_required:n = true ,
5388     fill .tl_set:N = \l_@@_fill_tl ,
5389     fill .value_required:n = true ,
5390     opacity .tl_set:N = \l_@@_opacity_tl ,
5391     opacity .value_required:n = true ,
5392     rowcolor .tl_set:N = \l_@@_fill_tl ,
5393     rowcolor .value_required:n = true ,
5394     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5395     rounded-corners .default:n = 4 pt ,
5396     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5397 }

5398 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5399 {
5400     \group_begin:
5401     \tl_clear:N \l_@@_fill_tl
5402     \tl_clear:N \l_@@_opacity_tl
5403     \tl_clear:N \l_@@_color_tl
5404     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5405     \dim_zero:N \l_@@_rounded_corners_dim
5406     \dim_zero:N \l_tmpa_dim
5407     \dim_zero:N \l_tmpb_dim
5408     \keys_set:nn { nicematrix / RowStyle } { #1 }

If the key fill (or its alias rowcolor) has been used.
5409     \tl_if_empty:NF \l_@@_fill_tl
5410     {
5411         \@@_add_opacity_to_fill:
5412         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5413         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5414     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5415     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5416     {
5417         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5418         - *
5419     }
5420     { \dim_use:N \l_@@_rounded_corners_dim }
5421 }
5422 }
5423 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5424     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5425     {
5426         \@@_put_in_row_style:e
5427         {
5428             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5429             {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5430         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5431         { \dim_use:N \l_tmpa_dim }
5432     }
5433 }
5434 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5435     \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5436     {
5437         \@@_put_in_row_style:e
5438         {
5439             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5440             {
5441                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5442                 { \dim_use:N \l_tmpb_dim }
5443             }
5444         }
5445     }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5446     \tl_if_empty:NF \l_@@_color_tl
5447     {
5448         \@@_put_in_row_style:e
5449         {
5450             \mode_leave_vertical:
5451             \@@_color:n { \l_@@_color_tl }
5452         }
5453     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5454     \bool_if:NT \l_@@_bold_row_style_bool
5455     {
5456         \@@_put_in_row_style:n
5457         {
5458             \exp_not:n
5459             {
5460                 \if_mode_math:
5461                 \c_math_toggle_token
5462                 \bfseries \boldmath
5463                 \c_math_toggle_token
5464             }
5465             \else:
5466             \bfseries \boldmath
5467             \fi:
5467         }

```

```

5468     }
5469   }
5470   \group_end:
5471   \g_@@_row_style_tl
5472   \ignorespaces
5473 }

```

The following commande must *not* be protected.

```

5474 \cs_new:Npn \@@_rounded_from_row:n #1
5475 {
5476   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5477   { \int_eval:n { #1 } - 1 }
5478   {
5479     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5480     - \exp_not:n { \int_use:N \c@jCol }
5481   }
5482   { \dim_use:N \l_@@_rounded_corners_dim }
5483 }

```

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).

```

5484 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5485 {

```

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.

```

5486   \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`.

```

5487   \str_if_in:nnF { #1 } { !! }
5488   {
5489     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5490     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5491   }
5492   \int_if_zero:nTF { \l_tmpa_int }
```

First, the case where the color is a *new* color (not in the sequence).

```
5493   {
5494     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5495     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5496   }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5497     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5498   }
5499   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5500   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5501   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5502   {
5503     \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5504     {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5505     \group_begin:
5506     \pgfsetcornersarced
5507     {
5508       \pgfpoint
5509       { \l_@@_tab_rounded_corners_dim }
5510       { \l_@@_tab_rounded_corners_dim }
5511     }
```

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.

```
5512     \bool_if:NTF \l_@@_hvlines_bool
5513     {
5514       \pgfpathrectanglecorners
5515       {
5516         \pgfpointadd
5517         { \@@_qpoint:n { row-1 } }
5518         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5519       }
5520     {
5521       \pgfpointadd
5522       {
5523         \@@_qpoint:n
5524         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5525       }
5526       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5527     }
5528   }
5529   {
5530     \pgfpathrectanglecorners
5531     { \@@_qpoint:n { row-1 } }
5532     {
5533       \pgfpointadd
5534       {
5535         \@@_qpoint:n
5536         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5537       }
5538     }
```

```

5538         { \pgfpoint \c_zero_dim \arrayrulewidth }
5539     }
5540 }
5541 \pgfusepath { clip }
5542 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5543 }
5544 }

```

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`).

```

5545 \cs_new_protected:Npn \@@_actually_color:
5546 {
5547     \pgfpicture
5548     \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.

```

5549     \@@_clip_with_rounded_corners:
5550     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5551     {
5552         \int_compare:nNnTF { ##1 } = { \c_one_int }
5553         {
5554             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5555             \use:c { g_@@_color _ 1 _tl }
5556             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5557         }
5558         {
5559             \begin { pgfscope }
5560                 \@@_color_opacity: ##2
5561                 \use:c { g_@@_color _ ##1 _tl }
5562                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5563                 \pgfusepath { fill }
5564             \end { pgfscope }
5565         }
5566     }
5567 \endpgfpicture
5568 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5569 \cs_new_protected:Npn \@@_color_opacity:
5570 {
5571     \peek_meaning:NTF [
5572         { \@@_color_opacity:w }
5573         { \@@_color_opacity:w [ ] }
5574     }

```

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.

```

5575 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5576 {
5577     \tl_clear:N \l_tmpa_tl
5578     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5579     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5580     \tl_if_empty:NTF \l_tmpb_tl
5581     { \@declaredcolor }
5582     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5583 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5584 \keys_define:nn { nicematrix / color-opacity }
5585   {
5586     opacity .tl_set:N          = \l_tmpa_tl ,
5587     opacity .value_required:n = true
5588   }

```

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

```

5589 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5590   {
5591     \def \l_@@_rows_tl { #1 }
5592     \def \l_@@_cols_tl { #2 }
5593     \@@_cartesian_path:
5594   }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5595 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5596   {
5597     \tl_if_blank:nF { #2 }
5598     {
5599       \@@_add_to_colors_seq:en
5600       { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5601       { \@@_cartesian_color:nn { #3 } { - } }
5602     }
5603   }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5604 \NewDocumentCommand \@@_columncolor { 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 is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5613 \NewDocumentCommand \@@_rectanglecolor { 0 { } m 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       { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5620     }
5621   }

```

The last argument is the radius of the corners of the rectangle.

```

5622 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m 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 } { #5 } }
5629     }
5630   }

```

The last argument is the radius of the corners of the rectangle.

```

5631 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5632 {
5633   \@@_cut_on_hyphen:w #1 \q_stop
5634   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5635   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5636   \@@_cut_on_hyphen:w #2 \q_stop
5637   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5638   \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`.

```

5639   \@@_cartesian_path:n { #3 }
5640 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5641 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5642 {
5643   \clist_map_inline:nn { #3 }
5644     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5645 }

```

```

5646 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5647 {
5648   \int_step_inline:nn { \c@iRow }
5649     {
5650       \int_step_inline:nn { \c@jCol }
5651         {
5652           \int_if_even:nTF { #####1 + ##1 }
5653             { \@@_cellcolor [ #1 ] { #2 } }
5654             { \@@_cellcolor [ #1 ] { #3 } }
5655           { ##1 - #####1 }
5656         }
5657     }
5658 }

```

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”.

```

5659 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5660 {
5661   \@@_rectanglecolor [ #1 ] { #2 }
5662     { 1 - 1 }
5663     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5664 }

```

```

5665 \keys_define:nn { nicematrix / rowcolors }
5666 {
5667   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5668   respect-blocks .default:n = true ,
5669   cols .tl_set:N = \l_@@_cols_tl ,
5670   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5671   restart .default:n = true ,
5672   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5673 }

```

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*.

```
5674 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5675 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
5676 \group_begin:
5677 \seq_clear_new:N \l_@@_colors_seq
5678 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5679 \tl_clear_new:N \l_@@_cols_tl
5680 \tl_set:Nn \l_@@_cols_tl { - }
5681 \keys_set:mn { 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).

```
5682 \int_zero_new:N \l_@@_color_int
5683 \int_set_eq:NN \l_@@_color_int \c_one_int
5684 \bool_if:NT \l_@@_respect_blocks_bool
5685 {
```

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`).

```
5686 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5687 \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5688 { \@@_not_in_exterior_p:nnnnn ##1 }
5689 }
5690 \pgfpicture
5691 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5692 \clist_map_inline:nn { #2 }
5693 {
5694 \tl_set:Nn \l_tmpa_tl { ##1 }
5695 \tl_if_in:NnTF \l_tmpa_tl { - }
5696 { \@@_cut_on_hyphen:w ##1 \q_stop }
5697 { \tl_set:No \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.

```
5698 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5699 \int_set:Nn \l_@@_color_int
5700 { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5701 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5702 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5703 {
```

We will compute in `l_tmpb_int` the last row of the "block".

```
5704 \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).

```
5705 \bool_if:NT \l_@@_respect_blocks_bool
5706 {
5707 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5708 { \@@_intersect_our_row_p:nnnnn #####1 }
5709 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }
```

Now, the last row of the block is computed in `l_tmpb_int`.

```
5710 }
5711 \tl_set:Ne \l_@@_rows_tl
5712 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5713     \tl_set:Nc \l_@@_color_tl
5714     {
5715         \@@_color_index:n
5716         {
5717             \int_mod:nn
5718             { \l_@@_color_int - 1 }
5719             { \seq_count:N \l_@@_colors_seq }
5720             + 1
5721         }
5722     }
5723     \tl_if_empty:NF \l_@@_color_tl
5724     {
5725         \@@_add_to_colors_seq:ee
5726         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5727         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5728     }
5729     \int_incr:N \l_@@_color_int
5730     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5731 }
5732 }
5733 \endpgfpicture
5734 \group_end:
5735 }

```

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.

```

5736 \cs_new:Npn \@@_color_index:n #1
5737 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5738     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5739     { \@@_color_index:n { #1 - 1 } }
5740     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5741 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5742 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5743 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5744 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5745 {
5746     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5747     { \int_set:Nn \l_tmpb_int { #3 } }
5748 }

```

```

5749 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5750 {
5751     \int_if_zero:nTF { #4 }
5752     { \prg_return_false: }
5753     {
5754         \int_compare:nNnTF { #2 } > { \c@jCol }
5755         { \prg_return_false: }
5756         { \prg_return_true: }
5757     }
5758 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5759 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5760 {
5761   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5762     { \prg_return_false: }
5763     {
5764       \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5765         { \prg_return_false: }
5766         { \prg_return_true: }
5767     }
5768 }

```

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).

```

5769 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5770 {
5771   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5772     {
5773       \bool_if:NTF \l_@@_nocolor_used_bool
5774         { \@@_cartesian_path_normal_ii: }
5775         {
5776           \clist_if_empty:NTF \l_@@_corners_cells_clist
5777             { \@@_cartesian_path_normal_i:n { #1 } }
5778             { \@@_cartesian_path_normal_ii: }
5779         }
5780     }
5781   { \@@_cartesian_path_normal_i:n { #1 } }
5782 }

```

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.

```

5783 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5784 {
5785   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5786   \clist_map_inline:Nn \l_@@_cols_tl
5787   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

5788     \def \l_tmpa_tl { ##1 }
5789     \tl_if_in:NnTF \l_tmpa_tl { - }
5790       { \@@_cut_on_hyphen:w ##1 \q_stop }
5791       { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5792     \tl_if_empty:NTF \l_tmpa_tl
5793       { \def \l_tmpa_tl { 1 } }
5794       {
5795         \str_if_eq:eeT \l_tmpa_tl { * }
5796         { \def \l_tmpa_tl { 1 } }
5797       }
5798     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5799       { \@@_error:n { Invalid~col~number } }
5800     \tl_if_empty:NTF \l_tmpb_tl
5801       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5802       {
5803         \str_if_eq:eeT \l_tmpb_tl { * }
5804         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5805       }

```

```

5806 \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5807 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5808 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5809 \@@_qpoint:n { col - \l_tmpa_tl }
5810 \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5811 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5812 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5813 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5814 \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.

```

5815 \clist_map_inline:Nn \l_@@_rows_tl
5816 {
5817   \def \l_tmpa_tl { #####1 }
5818   \tl_if_in:NnTF \l_tmpa_tl { - }
5819   { \@@_cut_on_hyphen:w #####1 \q_stop }
5820   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5821   \tl_if_empty:NnTF \l_tmpa_tl
5822   { \def \l_tmpa_tl { 1 } }
5823   {
5824     \str_if_eq:eeT \l_tmpa_tl { * }
5825     { \def \l_tmpa_tl { 1 } }
5826   }
5827   \tl_if_empty:NnTF \l_tmpb_tl
5828   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5829   {
5830     \str_if_eq:eeT \l_tmpb_tl { * }
5831     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5832   }
5833   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5834   { \@@_error:n { Invalid-row-number } }
5835   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5836   { \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`.

```

5837 \cs_if_exist:cF
5838 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5839 {
5840   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5841   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5842   \@@_qpoint:n { row - \l_tmpa_tl }
5843   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5844   \pgfpathrectanglecorners
5845   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5846   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5847 }
5848 }
5849 }
5850 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5851 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5852 {
5853   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5854   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5855 \clist_map_inline:Nn \l_@@_cols_tl
5856 {
5857   \@@_qpoint:n { col - ##1 }
5858   \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5859   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }

```

```

5860     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5861     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5862     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5863     \clist_map_inline:Nn \l_@@_rows_tl
5864     {
5865         \@@_if_in_corner:nF { #####1 - ##1 }
5866         {
5867             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5868             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5869             \@@_qpoint:n { row - #####1 }
5870             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5871             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5872             {
5873                 \pgfpathrectanglecorners
5874                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5875                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5876             }
5877         }
5878     }
5879 }
5880 }

```

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`).

```

5881 \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

```

5882 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5883 {
5884     \bool_set_true:N \l_@@_nocolor_used_bool
5885     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5886     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5887     \clist_map_inline:Nn \l_@@_rows_tl
5888     {
5889         \clist_map_inline:Nn \l_@@_cols_tl
5890         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5891     }
5892 }

```

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.

```

5893 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5894 {
5895     \clist_set_eq:NN \l_tmpa_clist #1
5896     \clist_clear:N #1
5897     \clist_map_inline:Nn \l_tmpa_clist
5898     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5899     \def \l_tmpa_tl { ##1 }
5900     \tl_if_in:NnTF \l_tmpa_tl { - }
5901     { \@@_cut_on_hyphen:w ##1 \q_stop }
5902     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5903     \bool_lazy_or:nnT
5904     { \str_if_eq_p:ee \l_tmpa_tl { * } }

```

```

5905     { \tl_if_blank_p:o \l_tmpa_tl }
5906     { \def \l_tmpa_tl { 1 } }
5907     \bool_lazy_or:nnT
5908     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5909     { \tl_if_blank_p:o \l_tmpb_tl }
5910     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5911     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5912     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5913     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5914     { \clist_put_right:Nn #1 { #####1 } }
5915   }
5916 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5917 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5918 {
5919   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5920   {

```

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`).

```

5921     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5922     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5923   }
5924   \ignorespaces
5925 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5926 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5927 {
5928   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5929   {
5930     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5931     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5932     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5933   }
5934   \ignorespaces
5935 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by currying).

```

5936 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5937 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5938 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5939 {

```

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`.

```

5940   \seq_gclear:N \g_tmpa_seq
5941   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5942   { \@@_rowlistcolors_tabular:nnnn #1 }
5943   \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`.

```

5944   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5945   {
5946     { \int_use:N \c@iRow }
5947     { \exp_not:n { #1 } }
5948     { \exp_not:n { #2 } }
5949     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5950   }
5951   \ignorespaces
5952 }

```

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`).

```

5953 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5954 {
5955   \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).

```

5956   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5957   {
5958     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5959     {
5960       \@@_rowlistcolors
5961       [ \exp_not:n { #2 } ]
5962       { #1 - \int_eval:n { \c@iRow - 1 } }
5963       { \exp_not:n { #3 } }
5964       [ \exp_not:n { #4 } ]
5965     }
5966   }
5967 }

```

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.

```

5968 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5969 {
5970   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5971   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5972   \seq_gclear:N \g_@@_rowlistcolors_seq
5973 }

```

```

5974 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5975 {
5976   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5977   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5978 }

```

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.

```

5979 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5980 {

```

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).

```
5981 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
5982 {
```

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).

```
5983 \tl_gput_left:Nc \g_@@_pre_code_before_tl
5984 {
5985 \exp_not:N \columncolor [ #1 ]
5986 { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5987 }
5988 }
5989 }

5990 \cs_new_protected:Npn \@@_EmptyColumn:n #1
5991 {
5992 \clist_map_inline:nn { #1 }
5993 {
5994 \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5995 { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
5996 \columncolor { nocolor } { ##1 }
5997 }
5998 }

5999 \cs_new_protected:Npn \@@_EmptyRow:n #1
6000 {
6001 \clist_map_inline:nn { #1 }
6002 {
6003 \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6004 { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6005 \rowcolor { nocolor } { ##1 }
6006 }
6007 }
```

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`.

```
6008 \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`.

```
6009 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6010 {
6011 \int_if_zero:nTF { \l_@@_first_col_int }
6012 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6013 {
```



```

6014     \int_if_zero:nTF { \c@jCol }
6015     {
6016         \int_compare:nNnF { \c@iRow } = { -1 }
6017         {
6018             \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6019             { #1 }
6020         }
6021     }
6022     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6023 }
6024 }

```

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.

```

6025 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6026 {
6027     \int_if_zero:nF { \c@iRow }
6028     {
6029         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6030         {
6031             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6032             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6033         }
6034     }
6035 }

```

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`.

```

6036 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6037 {
6038     \IfPackageLoadedTF { tikz }
6039     {
6040         \IfPackageLoadedTF { booktabs }
6041         { #2 }
6042         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6043     }
6044     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6045 }
6046 \NewExpandableDocumentCommand { \@@_TopRule } { }
6047 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6048 \cs_new:Npn \@@_TopRule_i:
6049 {
6050     \noalign \bgroup
6051     \peek_meaning:NTF [
6052     { \@@_TopRule_ii: }
6053     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6054 }
6055 \NewDocumentCommand \@@_TopRule_ii: { o }
6056 {
6057     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6058     {
6059         \@@_hline:n
6060         {
6061             position = \int_eval:n { \c@iRow + 1 } ,
6062             tikz =
6063             {
6064                 line~width = #1 ,
6065                 yshift = 0.25 \arrayrulewidth ,

```

```

6066         shorten~< = - 0.5 \arrayrulewidth
6067     } ,
6068     total-width = #1
6069 }
6070 }
6071 \skip_vertical:n { \belowrulesep + #1 }
6072 \egroup
6073 }

6074 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6075 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6076 \cs_new:Npn \@@_BottomRule_i:
6077 {
6078     \noalign \bgroup
6079     \peek_meaning:NTF [
6080         { \@@_BottomRule_ii: }
6081         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6082     ]

6083 \NewDocumentCommand \@@_BottomRule_ii: { o }
6084 {
6085     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6086     {
6087         \@@_hline:n
6088         {
6089             position = \int_eval:n { \c@iRow + 1 } ,
6090             tikz =
6091             {
6092                 line~width = #1 ,
6093                 yshift = 0.25 \arrayrulewidth ,
6094                 shorten~< = - 0.5 \arrayrulewidth
6095             } ,
6096             total-width = #1 ,
6097         }
6098     }
6099     \skip_vertical:N \aboverulesep
6100     \@@_create_row_node_i:
6101     \skip_vertical:n { #1 }
6102     \egroup
6103 }

6104 \NewExpandableDocumentCommand { \@@_MidRule } { }
6105 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6106 \cs_new:Npn \@@_MidRule_i:
6107 {
6108     \noalign \bgroup
6109     \peek_meaning:NTF [
6110         { \@@_MidRule_ii: }
6111         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6112     ]

6113 \NewDocumentCommand \@@_MidRule_ii: { o }
6114 {
6115     \skip_vertical:N \aboverulesep
6116     \@@_create_row_node_i:
6117     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6118     {
6119         \@@_hline:n
6120         {
6121             position = \int_eval:n { \c@iRow + 1 } ,
6122             tikz =
6123             {
6124                 line~width = #1 ,
6125                 yshift = 0.25 \arrayrulewidth ,
6126                 shorten~< = - 0.5 \arrayrulewidth

```

```

6127         } ,
6128         total-width = #1 ,
6129     }
6130 }
6131 \skip_vertical:n { \belowrulesep + #1 }
6132 \egroup
6133 }

```

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.

```

6134 \keys_define:nn { nicematrix / Rules }
6135 {
6136     position .int_set:N = \l_@@_position_int ,
6137     position .value_required:n = true ,
6138     start .int_set:N = \l_@@_start_int ,
6139     end .code:n =
6140         \bool_lazy_or:nnTF
6141         { \tl_if_empty_p:n { #1 } }
6142         { \str_if_eq_p:ee { #1 } { last } }
6143         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6144         { \int_set:Nn \l_@@_end_int { #1 } }
6145 }

```

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.

```

6146 \keys_define:nn { nicematrix / RulesBis }
6147 {
6148     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6149     multiplicity .initial:n = 1 ,
6150     dotted .bool_set:N = \l_@@_dotted_bool ,
6151     dotted .initial:n = false ,
6152     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]`.

```

6153     color .code:n =
6154         \@@_set_CTarc:n { #1 }
6155         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6156     color .value_required:n = true ,
6157     sep-color .code:n = \@@_set_CTrsc:n { #1 } ,
6158     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.

```

6159     tikz .code:n =
6160         \IfPackageLoadedTF { tikz }
6161         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6162         { \@@_error:n { tikz-without-tikz } } ,
6163     tikz .value_required:n = true ,
6164     total-width .dim_set:N = \l_@@_rule_width_dim ,

```

```

6165     total-width .value_required:n = true ,
6166     width .meta:n = { total-width = #1 } ,
6167     unknown .code:n = \@_error:n { Unknow~key~for~RulesBis }
6168 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6169 \cs_new_protected:Npn \@_vline:n #1
6170 {

```

The group is for the options.

```

6171     \group_begin:
6172     \int_set_eq:NN \l_@@_end_int \c@iRow
6173     \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).

```

6174     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6175     \@_vline_i:
6176     \group_end:
6177 }

```

```

6178 \cs_new_protected:Npn \@_vline_i:
6179 {

```

`\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`.

```

6180     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6181     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6182     \l_tmpa_tl
6183     {

```

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.

```

6184     \bool_gset_true:N \g_tmpa_bool
6185     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6186     { \@_test_vline_in_block:nnnn ##1 }
6187     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6188     { \@_test_vline_in_block:nnnn ##1 }
6189     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6190     { \@_test_vline_in_stroken_block:nnnn ##1 }
6191     \clist_if_empty:NF \l_@@_corners_clist { \@_test_in_corner_v: }
6192     \bool_if:NTF \g_tmpa_bool
6193     {
6194         \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.

```

6195         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6196     }
6197     {
6198         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6199         {
6200             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6201             \@_vline_ii:
6202             \int_zero:N \l_@@_local_start_int
6203         }
6204     }
6205 }
6206 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }

```

```

6207     {
6208     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6209     \@@_vline_ii:
6210     }
6211 }

6212 \cs_new_protected:Npn \@@_test_in_corner_v:
6213 {
6214     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6215     {
6216     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6217     { \bool_set_false:N \g_tmpa_bool }
6218     }
6219     {
6220     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6221     {
6222     \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6223     { \bool_set_false:N \g_tmpa_bool }
6224     {
6225     \@@_if_in_corner:nT
6226     { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6227     { \bool_set_false:N \g_tmpa_bool }
6228     }
6229     }
6230     }
6231 }

6232 \cs_new_protected:Npn \@@_vline_ii:
6233 {
6234     \tl_clear:N \l_@@_tikz_rule_tl
6235     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6236     \bool_if:NTF \l_@@_dotted_bool
6237     { \@@_vline_iv: }
6238     {
6239     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6240     { \@@_vline_iii: }
6241     { \@@_vline_v: }
6242     }
6243 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6244 \cs_new_protected:Npn \@@_vline_iii:
6245 {
6246     \pgfpicture
6247     \pgfrememberpicturepositiononpagetrue
6248     \pgf@relevantforpicturesizefalse
6249     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6250     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6251     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6252     \dim_set:Nn \l_tmpb_dim
6253     {
6254     \pgf@x
6255     - 0.5 \l_@@_rule_width_dim
6256     +
6257     ( \arrayrulewidth * \l_@@_multiplicity_int
6258     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6259     }
6260     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6261     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6262     \bool_lazy_all:nT
6263     {

```

```

6264     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6265     { \cs_if_exist_p:N \CT@drsc@ }
6266     { ! \tl_if_blank_p:o \CT@drsc@ }
6267   }
6268   {
6269     \group_begin:
6270     \CT@drsc@
6271     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6272     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6273     \dim_set:Nn \l_@@_tmpd_dim
6274       {
6275         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6276         * ( \l_@@_multiplicity_int - 1 )
6277       }
6278     \pgfpathrectanglecorners
6279       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6280       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6281     \pgfusepath { fill }
6282     \group_end:
6283   }
6284   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6285   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6286   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6287     {
6288       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6289       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6290       \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6291       \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6292     }
6293   \CT@arc@
6294   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6295   \pgfsetrectcap
6296   \pgfusepathqstroke
6297   \endpgfpicture
6298 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6299 \cs_new_protected:Npn \@@_vline_iv:
6300   {
6301     \pgfpicture
6302     \pgfrememberpicturepositiononpagetrue
6303     \pgf@relevantforpicturesizefalse
6304     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6305     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6306     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6307     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6308     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6309     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6310     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6311     \CT@arc@
6312     \@@_draw_line:
6313     \endpgfpicture
6314   }

```

The following code is for the case when the user uses the key `tikz`.

```

6315 \cs_new_protected:Npn \@@_vline_v:
6316   {
6317     \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).

```

6318 \CT@arc@
6319 \tl_if_empty:NF \l_@@_rule_color_tl
6320 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6321 \pgfrememberpicturepositiononpagetrue
6322 \pgf@relevantforpicturesizefalse
6323 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6324 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6325 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6326 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6327 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6328 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6329 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6330 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6331 ( \l_tmpb_dim , \l_tmpa_dim ) --
6332 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6333 \end { tikzpicture }
6334 }

```

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).

```

6335 \cs_new_protected:Npn \@@_draw_vlines:
6336 {
6337   \int_step_inline:nnn
6338   {
6339     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6340     { 2 }
6341     { 1 }
6342   }
6343   {
6344     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6345     { \c@jCol }
6346     { \int_eval:n { \c@jCol + 1 } }
6347   }
6348   {
6349     \str_if_eq:eeF \l_@@_vlines_clist { all }
6350     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6351     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6352   }
6353 }

```

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}`.

```

6354 \cs_new_protected:Npn \@@_hline:n #1
6355 {

```

The group is for the options.

```

6356   \group_begin:
6357   \int_set_eq:NN \l_@@_end_int \c@jCol
6358   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6359   \@@_hline_i:
6360   \group_end:
6361 }

6362 \cs_new_protected:Npn \@@_hline_i:
6363 {
6364   % \int_zero:N \l_@@_local_start_int
6365   % \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`.

```

6366     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6367     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6368     \l_tmpb_tl
6369     {

```

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.

```

6370     \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6371     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6372     { \@@_test_hline_in_block:nnnnn ##1 }

6373     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6374     { \@@_test_hline_in_block:nnnnn ##1 }

6375     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6376     { \@@_test_hline_in_stroken_block:nnnn ##1 }

6377     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6378     \bool_if:NTF \g_tmpa_bool
6379     {
6380         \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.

```

6381         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6382     }
6383     {
6384         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6385     {
6386         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6387         \@@_hline_ii:
6388         \int_zero:N \l_@@_local_start_int
6389     }
6390     }
6391 }
6392 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6393 {
6394     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6395     \@@_hline_ii:
6396 }
6397 }

```

```

6398 \cs_new_protected:Npn \@@_test_in_corner_h:
6399 {
6400     \int_compare:nNnTF { \l_tmpa_tl } = { \c@iRow + 1 }
6401     {
6402         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6403         { \bool_set_false:N \g_tmpa_bool }
6404     }
6405     {
6406         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6407         {
6408             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6409             { \bool_set_false:N \g_tmpa_bool }
6410             {
6411                 \@@_if_in_corner:nT
6412                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6413                 { \bool_set_false:N \g_tmpa_bool }
6414             }
6415         }
6416     }
6417 }

```



```

6418 \cs_new_protected:Npn \@@_hline_ii:
6419 {
6420   \tl_clear:N \l_@@_tikz_rule_tl
6421   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6422   \bool_if:NTF \l_@@_dotted_bool
6423     { \@@_hline_iv: }
6424     {
6425       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6426         { \@@_hline_iii: }
6427         { \@@_hline_v: }
6428     }
6429 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6430 \cs_new_protected:Npn \@@_hline_iii:
6431 {
6432   \pgfpicture
6433   \pgfrememberpicturepositiononpagetrue
6434   \pgf@relevantforpicturesizefalse
6435   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6436   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6437   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6438   \dim_set:Nn \l_tmpb_dim
6439     {
6440       \pgf@y
6441       - 0.5 \l_@@_rule_width_dim
6442       +
6443       ( \arrayrulewidth * \l_@@_multiplicity_int
6444         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6445     }
6446   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6447   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6448   \bool_lazy_all:nT
6449     {
6450       { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6451       { \cs_if_exist_p:N \CT@drsc@ }
6452       { ! \tl_if_blank_p:o \CT@drsc@ }
6453     }
6454     {
6455       \group_begin:
6456       \CT@drsc@
6457       \dim_set:Nn \l_@@_tmpd_dim
6458         {
6459           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6460           * ( \l_@@_multiplicity_int - 1 )
6461         }
6462       \pgfpathrectanglecorners
6463         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6464         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6465       \pgfusepathqfill
6466       \group_end:
6467     }
6468   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6469   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6470   \prg_replicate:mn { \l_@@_multiplicity_int - 1 }
6471     {
6472       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6473       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6474       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6475       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6476     }
6477   \CT@arc@
6478   \pgfsetlinewidth { 1.1 \arrayrulewidth }

```

```

6479 \pgfsetrectcap
6480 \pgfusepathqstroke
6481 \endpgfpicture
6482 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6483 \cs_new_protected:Npn \@@_hline_iv:
6484 {
6485 \pgfpicture
6486 \pgfrememberpicturepositiononpagetrue
6487 \pgf@relevantforpicturesizefalse
6488 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6489 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6490 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6491 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6492 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6493 \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6494 {
6495 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6496 \bool_if:NF \g_@@_delims_bool
6497 { \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.

```

6498 \tl_if_eq:NnF \g_@@_left_delim_tl (
6499 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6500 )
6501 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6502 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6503 \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }
6504 {
6505 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6506 \bool_if:NF \g_@@_delims_bool
6507 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6508 \tl_if_eq:NnF \g_@@_right_delim_tl )
6509 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6510 }
6511 \CT@arc@
6512 \@@_draw_line:
6513 \endpgfpicture
6514 }

```

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`).

```

6515 \cs_new_protected:Npn \@@_hline_v:

```

```

6516 {
6517   \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).

```

6518   \CT@arc@
6519   \tl_if_empty:NF \l_@@_rule_color_tl
6520     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6521   \pgfrememberpicturepositiononpagetrue
6522   \pgf@relevantforpicturesizefalse
6523   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6524   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6525   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6526   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6527   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6528   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6529   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6530   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6531     ( \l_tmpa_dim , \l_tmpb_dim ) --
6532     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6533   \end { tikzpicture }
6534 }

```

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).

```

6535 \cs_new_protected:Npn \@@_draw_hlines:
6536 {
6537   \int_step_inline:nnn
6538     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6539     {
6540       \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6541       { \c@iRow }
6542       { \int_eval:n { \c@iRow + 1 } }
6543     }
6544     {
6545       \str_if_eq:eeF \l_@@_hlines_clist { all }
6546       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6547       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6548     }
6549 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6550 \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.

```

6551 \cs_set:Npn \@@_Hline_i:n #1
6552 {
6553   \peek_remove_spaces:n
6554   {
6555     \peek_meaning:NTF \Hline
6556     { \@@_Hline_ii:nn { #1 + 1 } }
6557     { \@@_Hline_iii:n { #1 } }
6558   }
6559 }
6560 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6561 \cs_set:Npn \@@_Hline_iii:n #1
6562 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

```

```

6563 \cs_set_protected:Npn \@@_Hline_iv:n #1 #2
6564 {
6565   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6566   \skip_vertical:N \l_@@_rule_width_dim
6567   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6568   {
6569     \@@_hline:n
6570     {
6571       multiplicity = #1 ,
6572       position = \int_eval:n { \c@iRow + 1 } ,
6573       total-width = \dim_use:N \l_@@_rule_width_dim ,
6574       #2
6575     }
6576   }
6577   \egroup
6578 }

```

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`).

```

6579 \cs_new_protected:Npn \@@_custom_line:n #1
6580 {
6581   \str_clear_new:N \l_@@_command_str
6582   \str_clear_new:N \l_@@_ccommand_str
6583   \str_clear_new:N \l_@@_letter_str
6584   \tl_clear_new:N \l_@@_other_keys_tl
6585   \keys_set_known:nnN { 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.

```

6586   \bool_lazy_all:nTF
6587   {
6588     { \str_if_empty_p:N \l_@@_letter_str }
6589     { \str_if_empty_p:N \l_@@_command_str }
6590     { \str_if_empty_p:N \l_@@_ccommand_str }
6591   }
6592   { \@@_error:n { No~letter~and~no~command } }
6593   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6594 }
6595 \keys_define:nn { nicematrix / custom-line }
6596 {
6597   letter .str_set:N = \l_@@_letter_str ,
6598   letter .value_required:n = true ,
6599   command .str_set:N = \l_@@_command_str ,
6600   command .value_required:n = true ,
6601   ccommand .str_set:N = \l_@@_ccommand_str ,
6602   ccommand .value_required:n = true ,
6603 }

```

```

6604 \cs_new_protected:Npn \@@_custom_line_i:n #1
6605 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6606   \bool_set_false:N \l_@@_tikz_rule_bool
6607   \bool_set_false:N \l_@@_dotted_rule_bool
6608   \bool_set_false:N \l_@@_color_bool

```

```

6609 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6610 \bool_if:NT \l_@@_tikz_rule_bool
6611 {
6612   \IfPackageLoadedF { tikz }
6613   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6614   \bool_if:NT \l_@@_color_bool
6615   { \@@_error:n { color-in-custom-line-with-tikz } }
6616 }
6617 \bool_if:NT \l_@@_dotted_rule_bool
6618 {
6619   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6620   { \@@_error:n { key-multiplicity-with-dotted } }
6621 }
6622 \str_if_empty:NF \l_@@_letter_str
6623 {
6624   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6625   { \@@_error:n { Several-letters } }
6626   {
6627     \tl_if_in:NoTF
6628     \c_@@_forbidden_letters_str
6629     \l_@@_letter_str
6630     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6631     {

```

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.

```

6632         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6633         { \@@_v_custom_line:n { #1 } }
6634     }
6635 }
6636 }
6637 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6638 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6639 }
6640 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6641 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6642 \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.

```

6643 \keys_define:nn { nicematrix / custom-line-bis }
6644 {
6645   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6646   multiplicity .initial:n = 1 ,
6647   multiplicity .value_required:n = true ,
6648   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6649   color .value_required:n = true ,
6650   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6651   tikz .value_required:n = true ,
6652   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6653   dotted .value_forbidden:n = true ,
6654   total-width .code:n = { } ,
6655   total-width .value_required:n = true ,
6656   width .code:n = { } ,
6657   width .value_required:n = true ,
6658   sep-color .code:n = { } ,
6659   sep-color .value_required:n = true ,
6660   unknown .code:n = \@@_error:n { Unknown-key-for-custom-line }
6661 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
6662 \bool_new:N \l_@@_dotted_rule_bool
6663 \bool_new:N \l_@@_tikz_rule_bool
6664 \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`.

```
6665 \keys_define:nn { nicematrix / custom-line-width }
6666 {
6667   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6668   multiplicity .initial:n = 1 ,
6669   multiplicity .value_required:n = true ,
6670   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6671   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6672                       \bool_set_true:N \l_@@_total_width_bool ,
6673   total-width .value_required:n = true ,
6674   width .meta:n = { total-width = #1 } ,
6675   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6676 }
```

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`).

```
6677 \cs_new_protected:Npn \@@_h_custom_line:n #1
6678 {
```

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`).

```
6679   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6680   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6681 }
```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```
6682 \cs_new_protected:Npn \@@_c_custom_line:n #1
6683 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6684   \exp_args:Nc \NewExpandableDocumentCommand
6685     { nicematrix - \l_@@_ccommand_str }
6686     { 0 { } m }
6687     {
6688       \noalign
6689       {
6690         \@@_compute_rule_width:n { #1 , ##1 }
6691         \skip_vertical:n { \l_@@_rule_width_dim }
6692         \clist_map_inline:nn
6693           { ##2 }
6694           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6695       }
6696     }
6697   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6698 }
```

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*.

```

6699 \cs_new_protected:Npn \@@_c_custom_line_i:n #1 #2
6700 {
6701   \tl_if_in:nnTF { #2 } { - }
6702   { \@@_cut_on_hyphen:w #2 \q_stop }
6703   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6704   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6705   {
6706     \@@_hline:n
6707     {
6708       #1 ,
6709       start = \l_tmpa_tl ,
6710       end = \l_tmpb_tl ,
6711       position = \int_eval:n { \c@iRow + 1 } ,
6712       total-width = \dim_use:N \l_@@_rule_width_dim
6713     }
6714   }
6715 }

6716 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6717 {
6718   \bool_set_false:N \l_@@_tikz_rule_bool
6719   \bool_set_false:N \l_@@_total_width_bool
6720   \bool_set_false:N \l_@@_dotted_rule_bool
6721   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6722   \bool_if:NF \l_@@_total_width_bool
6723   {
6724     \bool_if:NTF \l_@@_dotted_rule_bool
6725     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6726     {
6727       \bool_if:NF \l_@@_tikz_rule_bool
6728       {
6729         \dim_set:Nn \l_@@_rule_width_dim
6730         {
6731           \arrayrulewidth * \l_@@_multiplicity_int
6732           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6733         }
6734       }
6735     }
6736   }
6737 }

6738 \cs_new_protected:Npn \@@_v_custom_line:n #1
6739 {
6740   \@@_compute_rule_width:n { #1 }

6741   \tl_gput_right:Ne \g_@@_array_preamble_tl
6742   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6743   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6744   {
6745     \@@_vline:n
6746     {
6747       #1 ,
6748       position = \int_eval:n { \c@jCol + 1 } ,
6749       total-width = \dim_use:N \l_@@_rule_width_dim
6750     }
6751   }
6752   \@@_rec_preamble:n
6753 }

6754 \@@_custom_line:n
6755 { 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.

```
6756 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6757 {
6758   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6759   {
6760     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6761     {
6762       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6763       {
6764         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6765         { \bool_gset_false:N \g_tmpa_bool }
6766       }
6767     }
6768   }
6769 }
```

The same for vertical rules.

```
6770 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6771 {
6772   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6773   {
6774     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6775     {
6776       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6777       {
6778         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6779         { \bool_gset_false:N \g_tmpa_bool }
6780       }
6781     }
6782   }
6783 }

6784 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6785 {
6786   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6787   {
6788     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6789     {
6790       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6791       { \bool_gset_false:N \g_tmpa_bool }
6792       {
6793         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6794         { \bool_gset_false:N \g_tmpa_bool }
6795       }
6796     }
6797   }
6798 }

6799 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6800 {
6801   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6802   {
6803     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6804     {
6805       \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6806       { \bool_gset_false:N \g_tmpa_bool }
6807       {
6808         \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6809         { \bool_gset_false:N \g_tmpa_bool }
6810       }
6811     }
6812   }
6813 }
```



```

6811     }
6812   }
6813 }

```

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.

```

6814 \cs_new_protected:Npn \@@_compute_corners:
6815 {
6816   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6817   { \@@_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`).

```

6818   \clist_clear:N \l_@@_corners_cells_clist
6819   \clist_map_inline:Nn \l_@@_corners_cells_clist
6820   {
6821     \str_case:nnF { ##1 }
6822     {
6823       { NW }
6824       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6825       { NE }
6826       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6827       { SW }
6828       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6829       { SE }
6830       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6831     }
6832     { \@@_error:nn { bad~corner } { ##1 } }
6833   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6834   \clist_if_empty:NF \l_@@_corners_cells_clist
6835   {

```

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.

```

6836     \tl_gput_right:Ne \g_@@_aux_tl
6837     {
6838       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6839       { \l_@@_corners_cells_clist }
6840     }
6841   }
6842 }

```

```

6843 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6844 {
6845   \int_step_inline:nnn { #1 } { #3 }
6846   {
6847     \int_step_inline:nnn { #2 } { #4 }
6848     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6849   }
6850 }

```

```

6851 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6852 {
6853   \cs_if_exist:cTF
6854     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6855     { \prg_return_true: }
6856     { \prg_return_false: }
6857 }

```

“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.

```

6858 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6859 {

```

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.

```

6860   \bool_set_false:N \l_tmpa_bool
6861   \int_zero_new:N \l_@@_last_empty_row_int
6862   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6863   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6864   {
6865     \bool_lazy_or:nnTF
6866     {
6867       \cs_if_exist_p:c
6868         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6869     }
6870     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6871     { \bool_set_true:N \l_tmpa_bool }
6872   }
6873   \bool_if:NF \l_tmpa_bool
6874     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6875 }
6876 }

```

Now, you determine the last empty cell in the row of number 1.

```

6877   \bool_set_false:N \l_tmpa_bool
6878   \int_zero_new:N \l_@@_last_empty_column_int
6879   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6880   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6881   {
6882     \bool_lazy_or:nnTF
6883     {
6884       \cs_if_exist_p:c
6885         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6886     }
6887     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6888     { \bool_set_true:N \l_tmpa_bool }
6889   }
6890   \bool_if:NF \l_tmpa_bool
6891     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6892 }
6893 }

```

Now, we loop over the rows.

```
6894 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6895 {
```

We treat the row number ##1 with another loop.

```
6896 \bool_set_false:N \l_tmpa_bool
6897 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6898 {
6899 \bool_lazy_or:nnTF
6900 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6901 { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6902 { \bool_set_true:N \l_tmpa_bool }
6903 {
6904 \bool_if:NF \l_tmpa_bool
6905 {
6906 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6907 \clist_put_right:Nn
6908 \l_@@_corners_cells_clist
6909 { ##1 - #####1 }
6910 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6911 }
6912 }
6913 }
6914 }
6915 }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```
6916 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6917 \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:NeT \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.

```
6918 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6919 \keys_define:nn { nicematrix / NiceMatrixBlock }
6920 {
6921 auto-columns-width .code:n =
6922 {
6923 \bool_set_true:N \l_@@_block_auto_columns_width_bool
6924 \dim_gzero_new:N \g_@@_max_cell_width_dim
6925 \bool_set_true:N \l_@@_auto_columns_width_bool
6926 }
6927 }

6928 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6929 {
6930 \int_gincr:N \g_@@_NiceMatrixBlock_int
6931 \dim_zero:N \l_@@_columns_width_dim
6932 \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6933 \bool_if:NT \l_@@_block_auto_columns_width_bool
6934 {
6935 \cs_if_exist:cT
```

```

6936     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6937     {
6938       \dim_set:Nn \l_@@_columns_width_dim
6939       {
6940         \use:c
6941         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6942       }
6943     }
6944   }
6945 }

```

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`).

```

6946 {
6947   \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.

```

6948   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6949   {
6950     \bool_if:NT \l_@@_block_auto_columns_width_bool
6951     {
6952       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6953       \iow_shipout:Ne \@mainaux
6954       {
6955         \cs_gset:cpn
6956         { @@ _ 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.

```

6957         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6958       }
6959       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6960     }
6961   }
6962   \ignorespacesafterend
6963 }

```

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).

```

6964 \cs_new_protected:Npn \@@_create_extra_nodes:
6965 {
6966   \bool_if:nTF \l_@@_medium_nodes_bool
6967   {
6968     \bool_if:NTF \l_@@_no_cell_nodes_bool
6969     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6970     {
6971       \bool_if:NTF \l_@@_large_nodes_bool
6972       \@@_create_medium_and_large_nodes:
6973       \@@_create_medium_nodes:
6974     }
6975   }
6976   {
6977     \bool_if:NT \l_@@_large_nodes_bool
6978     {
6979       \bool_if:NTF \l_@@_no_cell_nodes_bool

```

```

6980         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6981         \@@_create_large_nodes:
6982     }
6983 }
6984 }

```

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`.

```

6985 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6986 {
6987     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6988     {
6989         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6990         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6991         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6992         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
6993     }
6994     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6995     {
6996         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
6997         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
6998         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
6999         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7000     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7001     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7002     {
7003         \int_step_variable:nnNn
7004         \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.

```

7005         {
7006             \cs_if_exist:cT
7007             { 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`.

```

7008         {
7009             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7010             \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7011             { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7012             \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7013             {
7014                 \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7015                 { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7016             }

```

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`.

```

7017     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7018     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7019     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } { \pgf@y } }
7020     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7021     {
7022         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7023         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } { \pgf@x } }
7024     }
7025 }
7026 }
7027 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7028 \int_step_variable:nNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7029 {
7030     \dim_compare:nNnT
7031     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7032     {
7033         \@@_qpoint:n { row - \@@_i: - base }
7034         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7035         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7036     }
7037 }
7038 \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7039 {
7040     \dim_compare:nNnT
7041     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7042     {
7043         \@@_qpoint:n { col - \@@_j: }
7044         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7045         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7046     }
7047 }
7048 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7049 \cs_new_protected:Npn \@@_create_medium_nodes:
7050 {
7051     \pgfpicture
7052     \pgfrememberpicturepositiononpagetrue
7053     \pgf@relevantforpicturesizefalse
7054     \@@_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”.

```

7055     \tl_set:Nn \l_@@_suffix_tl { -medium }
7056     \@@_create_nodes:
7057     \endpgfpicture
7058 }

```

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:`.

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7059 \cs_new_protected:Npn \@@_create_large_nodes:
7060 {
7061   \pgfpicture
7062     \pgfrememberpicturepositiononpagetrue
7063     \pgf@relevantforpicturesizefalse
7064     \@@_computations_for_medium_nodes:
7065     \@@_computations_for_large_nodes:
7066     \tl_set:Nn \l_@@_suffix_tl { - large }
7067     \@@_create_nodes:
7068   \endpgfpicture
7069 }
7070 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7071 {
7072   \pgfpicture
7073     \pgfrememberpicturepositiononpagetrue
7074     \pgf@relevantforpicturesizefalse
7075     \@@_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”.

```

7076     \tl_set:Nn \l_@@_suffix_tl { - medium }
7077     \@@_create_nodes:
7078     \@@_computations_for_large_nodes:
7079     \tl_set:Nn \l_@@_suffix_tl { - large }
7080     \@@_create_nodes:
7081   \endpgfpicture
7082 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

7083 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7084 {
7085   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7086   \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`.

```

7087   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7088   {
7089     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7090     {
7091       (
7092         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7093         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7094       )
7095       / 2
7096     }
7097     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7098     { l_@@_row _ \@@_i: _ min _ dim }
7099   }
7100   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7101   {
7102     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7103     {
7104       (
7105         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7106         \dim_use:c
7107         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7108       )
7109       / 2
7110     }
7111     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7112     { l_@@_column _ \@@_j: _ max _ dim }
7113   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7114   \dim_sub:cn
7115     { l_@@_column _ 1 _ min _ dim }
7116     \l_@@_left_margin_dim
7117   \dim_add:cn
7118     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7119     \l_@@_right_margin_dim
7120   }

```

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).

```

7121 \cs_new_protected:Npn \@@_create_nodes:
7122 {
7123   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7124   {
7125     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7126     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7127       \@@_pgf_rect_node:nnnnn
7128       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7129       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7130       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7131       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7132       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7133     \str_if_empty:NF \l_@@_name_str
7134     {
7135       \pgfnodealias
7136       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7137       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7138     }
7139   }
7140 }
7141 \int_step_inline:nn { \c@iRow }
7142 {
7143   \pgfnodealias
7144   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7145   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7146 }
7147 \int_step_inline:nn { \c@jCol }
7148 {
7149   \pgfnodealias
7150   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7151   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7152 }
7153 \pgfnodealias % added 2025-04-05
7154 { \@@_env: - last - last \l_@@_suffix_tl }
7155 { \@@_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 .

```

7156   \seq_map_pairwise_function:NNN
7157   \g_@@_multicolumn_cells_seq
7158   \g_@@_multicolumn_sizes_seq
7159   \@@_node_for_multicolumn:nn
7160 }

```



```

7161 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7162 {
7163   \cs_set_nopar:Npn \@@_i: { #1 }
7164   \cs_set_nopar:Npn \@@_j: { #2 }
7165 }

```

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”).

```

7166 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7167 {
7168   \@@_extract_coords_values: #1 \q_stop
7169   \@@_pgf_rect_node:nnnnn
7170   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7171   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7172   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7173   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7174   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7175   \str_if_empty:NF \l_@@_name_str
7176   {
7177     \pgfnodealias
7178     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7179     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7180   }
7181 }

```

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).

```

7182 \keys_define:mn { nicematrix / Block / FirstPass }
7183 {
7184   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7185     \bool_set_true:N \l_@@_p_block_bool ,
7186   j .value_forbidden:n = true ,
7187   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7188   l .value_forbidden:n = true ,
7189   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7190   r .value_forbidden:n = true ,
7191   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7192   c .value_forbidden:n = true ,
7193   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7194   L .value_forbidden:n = true ,
7195   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7196   R .value_forbidden:n = true ,
7197   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7198   C .value_forbidden:n = true ,
7199   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7200   t .value_forbidden:n = true ,
7201   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7202   T .value_forbidden:n = true ,
7203   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7204   b .value_forbidden:n = true ,
7205   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7206   B .value_forbidden:n = true ,

```

```

7207 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7208 m .value_forbidden:n = true ,
7209 v-center .meta:n = m ,
7210 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7211 p .value_forbidden:n = true ,
7212 color .code:n =
7213   \@@_color:n { #1 }
7214   \tl_set_rescan:Nnn
7215     \l_@@_draw_tl
7216     { \char_set_catcode_other:N ! }
7217     { #1 } ,
7218 color .value_required:n = true ,
7219 respect-arraystretch .code:n =
7220   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7221 respect-arraystretch .value_forbidden:n = true ,
7222 }

```

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.

```

7223 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7224 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7225 {

```

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).

```

7226   \tl_if_blank:nTF { #2 }
7227   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7228   {
7229     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7230     \@@_Block_i_czech:w \@@_Block_i:w
7231     #2 \q_stop
7232   }
7233   { #1 } { #3 } { #4 }
7234   \ignorespaces
7235 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7236 \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`.

```

7237 {
7238   \char_set_catcode_active:N -
7239   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7240 }

```

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.

```

7241 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7242 {

```

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).

```

7243   \bool_lazy_or:nnTF

```

```

7244 { \tl_if_blank_p:n { #1 } }
7245 { \str_if_eq_p:ee { * } { #1 } }
7246 { \int_set:Nn \l_tmpa_int { 100 } }
7247 { \int_set:Nn \l_tmpa_int { #1 } }
7248 \bool_lazy_or:nnTF
7249 { \tl_if_blank_p:n { #2 } }
7250 { \str_if_eq_p:ee { * } { #2 } }
7251 { \int_set:Nn \l_tmpb_int { 100 } }
7252 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7253 \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7254 {
7255   \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7256     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7257     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7258 }
7259 { \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.

```

7260 \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7261 \tl_set:Ne \l_tmpa_tl
7262 {
7263   { \int_use:N \c@iRow }
7264   { \int_use:N \c@jCol }
7265   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7266   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7267 }

```

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).

```

7268 \bool_set_false:N \l_tmpa_bool
7269 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7270 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7271 \bool_case:nF
7272 {
7273   \l_tmpa_bool { \@@_Block_vii:eennn }
7274   \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.

```

7275 \l_@@_X_bool { \@@_Block_v:eennn }
7276 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7277 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7278 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7279 }
7280 { \@@_Block_v:eennn }
7281 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7282 }

```

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.

```

7283 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7284 {
7285   \int_gincr:N \g_@@_block_box_int
7286   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7287   {
7288     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7289     {
7290       \@@_actually_diagbox:nnnnnn
7291       { \int_use:N \c@iRow }
7292       { \int_use:N \c@jCol }
7293       { \int_eval:n { \c@iRow + #1 - 1 } }
7294       { \int_eval:n { \c@jCol + #2 - 1 } }
7295       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7296       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7297     }
7298   }
7299   \box_gclear_new:c
7300   { 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!

```

7301   \hbox_gset:cn
7302   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7303   {

```

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`).

```

7304   \tl_if_empty:NTF \l_@@_color_tl
7305   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7306   { \@@_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`.

```

7307   \int_compare:nNnT { #1 } = { \c_one_int }
7308   {
7309     \int_if_zero:nTF { \c@iRow }
7310     {

```

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}$

```

```

7311         \cs_set_eq:NN \Block \@@_NullBlock:
7312         \l_@@_code_for_first_row_tl
7313     }
7314     {
7315         \int_compare:nNnT { \c{iRow} } = { \l_@@_last_row_int }
7316         {
7317             \cs_set_eq:NN \Block \@@_NullBlock:
7318             \l_@@_code_for_last_row_tl
7319         }
7320     }
7321     \g_@@_row_style_tl
7322 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7323     \@@_reset_arraystretch:
7324     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7325     #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.).

```

7326     \@@_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}`.

```

7327     \bool_if:NTF \l_@@_tabular_bool
7328     {
7329         \bool_lazy_all:nTF
7330         {
7331             { \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`.

```

7332     {
7333         ! \dim_compare_p:nNn
7334         { \l_@@_col_width_dim } < { \c_zero_dim }
7335     }
7336     { ! \g_@@_rotate_bool }
7337 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7338     {
7339         \use:e
7340         {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7341         \exp_not:N \begin { minipage }
7342         [ \str_lowercase:f \l_@@_vpos_block_str ]
7343         { \l_@@_col_width_dim }
7344         \str_case:on \l_@@_hpos_block_str

```

```

7345         { c \centering r \raggedleft l \raggedright }
7346     }
7347     #5
7348 \end { minipage }
7349 }

```

In the other cases, we use a `{tabular}`.

```

7350     {
7351     \bool_if:NT \c_@@_testphase_table_bool
7352     { \tagpdfsetup { table / tagging = presentation } }
7353     \use:e
7354     {
7355     \exp_not:N \begin { tabular }
7356     [ \str_lowercase:f \l_@@_vpos_block_str ]
7357     { @ { } \l_@@_hpos_block_str @ { } }
7358     }
7359     #5
7360     \end { tabular }
7361     }
7362 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7363     {
7364     \c_math_toggle_token
7365     \use:e
7366     {
7367     \exp_not:N \begin { array }
7368     [ \str_lowercase:f \l_@@_vpos_block_str ]
7369     { @ { } \l_@@_hpos_block_str @ { } }
7370     }
7371     #5
7372     \end { array }
7373     \c_math_toggle_token
7374     }
7375 }

```

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).

```

7376     \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.

```

7377     \int_compare:nNnT { #2 } = { \c_one_int }
7378     {
7379     \dim_gset:Nn \g_@@_blocks_wd_dim
7380     {
7381     \dim_max:nn
7382     { \g_@@_blocks_wd_dim }
7383     {
7384     \box_wd:c
7385     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7386     }
7387     }
7388 }

```

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.

```

7389     \bool_lazy_and:nnT
7390     { \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.

```

7391     { \str_if_empty_p:N \l_@@_vpos_block_str }
7392     {
7393       \dim_gset:Nn \g_@@_blocks_ht_dim
7394       {
7395         \dim_max:nn
7396         { \g_@@_blocks_ht_dim }
7397         {
7398           \box_ht:c
7399           { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7400         }
7401       }
7402     \dim_gset:Nn \g_@@_blocks_dp_dim
7403     {
7404       \dim_max:nn
7405       { \g_@@_blocks_dp_dim }
7406       {
7407         \box_dp:c
7408         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7409       }
7410     }
7411   }
7412 \seq_gput_right:Ne \g_@@_blocks_seq
7413 {
7414   \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.

```

7415     {
7416       \exp_not:n { #3 } ,
7417       \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7418     \bool_if:NT \g_@@_rotate_bool
7419     {
7420       \bool_if:NTF \g_@@_rotate_c_bool
7421       { m }
7422       {
7423         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7424         { T }
7425       }
7426     }
7427   }
7428   {
7429     \box_use_drop:c
7430     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7431   }
7432 }
7433 \bool_set_false:N \g_@@_rotate_c_bool
7434 }

```

```

7435 \cs_new:Npn \@@_adjust_hpos_rotate:
7436 {
7437   \bool_if:NT \g_@@_rotate_bool
7438   {
7439     \str_set:Ne \l_@@_hpos_block_str
7440     {
7441       \bool_if:NTF \g_@@_rotate_c_bool
7442       { c }
7443       {

```

```

7444         \str_case:onF \l_@@_vpos_block_str
7445             { b l B l t r T r }
7446             {
7447                 \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7448                     { r }
7449                     { l }
7450             }
7451         }
7452     }
7453 }
7454 }
7455 \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*.

```

7456 \cs_new_protected:Npn \@@_rotate_box_of_block:
7457 {
7458     \box_grotate:cn
7459     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7460     { 90 }
7461     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7462     {
7463         \vbox_gset_top:cn
7464         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7465         {
7466             \skip_vertical:n { 0.8 ex }
7467             \box_use:c
7468             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7469         }
7470     }
7471     \bool_if:NT \g_@@_rotate_c_bool
7472     {
7473         \hbox_gset:cn
7474         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7475         {
7476             \c_math_toggle_token
7477             \vcenter
7478             {
7479                 \box_use:c
7480                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7481             }
7482             \c_math_toggle_token
7483         }
7484     }
7485 }

```

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:nnnnn`).

`#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7486 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7487 {
7488     \seq_gput_right:Ne \g_@@_blocks_seq
7489     {
7490         \l_tmpa_tl
7491         { \exp_not:n { #3 } }
7492         {
7493             \bool_if:NTF \l_@@_tabular_bool
7494             {

```


7495

`\group_begin:`

The following command will be no-op when `respect-arraystretch` is in force.

7496

`\@@_reset_arraystretch:`

7497

`\exp_not:n`

7498

`{`

7499

`\dim_zero:N \extrarowheight`

7500

`#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).

7501

`\bool_if:NT \c_@@_testphase_table_bool`

7502

`{ \tag_stop:n { table } }`

7503

`\use:e`

7504

`{`

7505

`\exp_not:N \begin { tabular } [\l_@@_vpos_block_str]`

7506

`{ @ { } \l_@@_hpos_block_str @ { } }`

7507

`}`

7508

`#5`

7509

`\end { tabular }`

7510

`}`

7511

`\group_end:`

7512

`}`

When we are *not* in an environment `{NiceTabular}` (or similar).

7513

`{`

7514

`\group_begin:`

The following will be no-op when `respect-arraystretch` is in force.

7515

`\@@_reset_arraystretch:`

7516

`\exp_not:n`

7517

`{`

7518

`\dim_zero:N \extrarowheight`

7519

`#4`

7520

`\c_math_toggle_token`

7521

`\use:e`

7522

`{`

7523

`\exp_not:N \begin { array } [\l_@@_vpos_block_str]`

7524

`{ @ { } \l_@@_hpos_block_str @ { } }`

7525

`}`

7526

`#5`

7527

`\end { array }`

7528

`\c_math_toggle_token`

7529

`}`

7530

`\group_end:`

7531

`}`

7532

`}`

7533

`}`

7534

`}`

7535

`\cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }`

The following macro is for the case of a `\Block` which uses the key `p`.

7536

`\cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5`

7537

`{`

7538

`\seq_gput_right:Ne \g_@@_blocks_seq`

7539

`{`

7540

`\l_tmpa_tl`

7541

`{ \exp_not:n { #3 } }`

Here, the curly braces for the group are mandatory.

7542

`{ { \exp_not:n { #4 #5 } } }`

```

7543     }
7544   }
7545 \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`.

```

7546 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7547 {
7548   \seq_gput_right:Ne \g_@@_blocks_seq
7549   {
7550     \l_tmpa_tl
7551     { \exp_not:n { #3 } }
7552     { \exp_not:n { #4 #5 } }
7553   }
7554 }
7555 \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).

```

7556 \keys_define:nn { nicematrix / Block / SecondPass }
7557 {
7558   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7559   ampersand-in-blocks .default:n = true ,
7560   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7561   tikz .code:n =
7562     \IfPackageLoadedTF { tikz }
7563     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7564     { \@@_error:n { tikz~key~without~tikz } } ,
7565   tikz .value_required:n = true ,
7566   fill .code:n =
7567     \tl_set_rescan:Nnn
7568     \l_@@_fill_tl
7569     { \char_set_catcode_other:N ! }
7570     { #1 } ,
7571   fill .value_required:n = true ,
7572   opacity .tl_set:N = \l_@@_opacity_tl ,
7573   opacity .value_required:n = true ,
7574   draw .code:n =
7575     \tl_set_rescan:Nnn
7576     \l_@@_draw_tl
7577     { \char_set_catcode_other:N ! }
7578     { #1 } ,
7579   draw .default:n = default ,
7580   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7581   rounded-corners .default:n = 4 pt ,
7582   color .code:n =
7583     \@@_color:n { #1 }
7584     \tl_set_rescan:Nnn
7585     \l_@@_draw_tl
7586     { \char_set_catcode_other:N ! }
7587     { #1 } ,
7588   borders .clist_set:N = \l_@@_borders_clist ,
7589   borders .value_required:n = true ,
7590   hvlines .meta:n = { vlines , hlines } ,
7591   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7592   vlines .default:n = true ,
7593   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7594   hlines .default:n = true ,
7595   line-width .dim_set:N = \l_@@_line_width_dim ,
7596   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7597   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7598           \bool_set_true:N \l_@@_p_block_bool ,
7599   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7600   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7601   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7602   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7603           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7604   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7605           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7606   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7607           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7608   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7609   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7610   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7611   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7612   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7613   m .value_forbidden:n = true ,
7614   v-center .meta:n = m ,
7615   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7616   p .value_forbidden:n = true ,
7617   name .tl_set:N = \l_@@_block_name_str ,
7618   name .value_required:n = true ,
7619   name .initial:n = ,
7620   respect-arraystretch .code:n =
7621     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7622   respect-arraystretch .value_forbidden:n = true ,
7623   transparent .bool_set:N = \l_@@_transparent_bool ,
7624   transparent .default:n = true ,
7625   transparent .initial:n = false ,
7626   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7627 }

```

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.

```

7628 \cs_new_protected:Npn \@@_draw_blocks:
7629 {
7630   \bool_if:NTF \c_@@_recent_array_bool
7631     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7632     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7633   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7634 }
7635 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7636 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7637   \int_zero:N \l_@@_last_row_int
7638   \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”).

```

7639   \int_compare:nNnTF { #3 } > { 98 }
7640     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7641     { \int_set:Nn \l_@@_last_row_int { #3 } }
7642   \int_compare:nNnTF { #4 } > { 98 }
7643     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7644     { \int_set:Nn \l_@@_last_col_int { #4 } }

```

```

7645 \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7646 {
7647   \bool_lazy_and:nnTF
7648     { \l_@@_preamble_bool }
7649     {
7650       \int_compare_p:n
7651         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7652     }
7653     {
7654       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7655       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7656       \@@_msg_redirect_name:nn { columns-not-used } { none }
7657     }
7658     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7659   }
7660   {
7661     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7662     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7663     {
7664       \@@_Block_v:nneenn
7665         { #1 }
7666         { #2 }
7667         { \int_use:N \l_@@_last_row_int }
7668         { \int_use:N \l_@@_last_col_int }
7669         { #5 }
7670         { #6 }
7671     }
7672   }
7673 }

```

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

```

7674 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7675 {

```

The group is for the keys.

```

7676   \group_begin:
7677   \int_compare:nNnT { #1 } = { #3 }
7678     { \str_set:Nn \l_@@_vpos_block_str { t } }
7679   \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7680   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7681   \bool_lazy_and:nnT
7682     { \l_@@_vlines_block_bool }
7683     { ! \l_@@_ampersand_bool }
7684   {
7685     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7686     {
7687       \@@_vlines_block:nnn
7688         { \exp_not:n { #5 } }
7689         { #1 - #2 }
7690         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7691     }
7692   }
7693   \bool_if:NT \l_@@_hlines_block_bool
7694   {
7695     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7696     {
7697       \@@_hlines_block:nnn
7698       { \exp_not:n { #5 } }

```

```

7699         { #1 - #2 }
7700         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7701     }
7702 }
7703 \bool_if:NF \l_@@_transparent_bool
7704 {
7705     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7706     {

```

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).

```

7707     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7708     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7709     }
7710 }

7711 \tl_if_empty:NF \l_@@_draw_tl
7712 {
7713     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7714     { \@@_error:n { hlines~with~color } }
7715     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7716     {
7717         \@@_stroke_block:nnn

```

#5 are the options

```

7718         { \exp_not:n { #5 } }
7719         { #1 - #2 }
7720         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7721     }
7722     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7723     { { #1 } { #2 } { #3 } { #4 } }
7724 }

7725 \clist_if_empty:NF \l_@@_borders_clist
7726 {
7727     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7728     {
7729         \@@_stroke_borders_block:nnn
7730         { \exp_not:n { #5 } }
7731         { #1 - #2 }
7732         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7733     }
7734 }

7735 \tl_if_empty:NF \l_@@_fill_tl
7736 {
7737     \@@_add_opacity_to_fill:
7738     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7739     {
7740         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7741         { #1 - #2 }
7742         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7743         { \dim_use:N \l_@@_rounded_corners_dim }
7744     }
7745 }

7746 \seq_if_empty:NF \l_@@_tikz_seq
7747 {
7748     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7749     {
7750         \@@_block_tikz:nnnnn
7751         { \seq_use:Nn \l_@@_tikz_seq { , } }
7752         { #1 }
7753         { #2 }
7754         { \int_use:N \l_@@_last_row_int }
7755         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7756     }
7757   }

7758   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7759   {
7760     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7761     {
7762       \@@_actually_diagbox:nnnnn
7763       { #1 }
7764       { #2 }
7765       { \int_use:N \l_@@_last_row_int }
7766       { \int_use:N \l_@@_last_col_int }
7767       { \exp_not:n { ##1 } }
7768       { \exp_not:n { ##2 } }
7769     }
7770   }

```

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.

```

7771   \pgfpicture
7772   \pgfrememberpicturepositiononpagetrue
7773   \pgf@relevantforpicturesizefalse
7774   \@@_qpoint:n { row - #1 }
7775   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7776   \@@_qpoint:n { col - #2 }
7777   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7778   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7779   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7780   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7781   \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.

```

7782   \@@_pgf_rect_node:nnnnn
7783   { \@@_env: - #1 - #2 - block }
7784   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7785   \str_if_empty:NF \l_@@_block_name_str
7786   {
7787     \pgfnodealias
7788     { \@@_env: - \l_@@_block_name_str }
7789     { \@@_env: - #1 - #2 - block }
7790     \str_if_empty:NF \l_@@_name_str

```

```

7791     {
7792         \pgfnodealias
7793         { \l_@@_name_str - \l_@@_block_name_str }
7794         { \@@_env: - #1 - #2 - block }
7795     }
7796 }

```

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.

```

7797     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7798     {
7799         \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.

```

7800         \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7801         {

```

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.

```

7802             \cs_if_exist:cT
7803             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7804             {
7805                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7806                 {
7807                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7808                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7809                 }
7810             }
7811         }

```

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.

```

7812         \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7813         {
7814             \@@_qpoint:n { col - #2 }
7815             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7816         }
7817     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7818     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7819     {
7820         \cs_if_exist:cT
7821         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7822         {
7823             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7824             {
7825                 \pgfpointanchor
7826                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7827                 { east }
7828                 \dim_set:Nn \l_@@_tmpd_dim
7829                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7830             }
7831         }
7832     }
7833     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7834     {
7835         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7836         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7837     }
7838     \@@_pgf_rect_node:nnnnn
7839     { \@@_env: - #1 - #2 - block - short }
7840     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim

```

7841 }

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.

```

7842 \bool_if:NT \l_@@_medium_nodes_bool
7843 {
7844   \@@_pgf_rect_node:nnn
7845   { \@@_env: - #1 - #2 - block - medium }
7846   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7847   {
7848     \pgfpointanchor
7849     { \@@_env:
7850       - \int_use:N \l_@@_last_row_int
7851       - \int_use:N \l_@@_last_col_int - medium
7852     }
7853     { south-east }
7854   }
7855 }
7856 \endpgfpicture

```

```

7857 \bool_if:NTF \l_@@_ampersand_bool
7858 {
7859   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7860   \int_zero_new:N \l_@@_split_int
7861   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7862   \pgfpicture
7863   \pgfrememberpicturepositiononpagetrue
7864   \pgf@relevantforpicturesizefalse
7865
7866   \@@_qpoint:n { row - #1 }
7867   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7868   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7869   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7870   \@@_qpoint:n { col - #2 }
7871   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7872   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7873   \dim_set:Nn \l_tmpb_dim
7874     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7875   \bool_lazy_or:nnT
7876     { \l_@@_vlines_block_bool }
7877     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7878     {
7879       \int_step_inline:nn { \l_@@_split_int - 1 }
7880       {
7881         \pgfpathmoveto
7882         {
7883           \pgfpoint
7884             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7885             \l_@@_tmpc_dim
7886         }
7887         \pgfpathlineto
7888         {
7889           \pgfpoint
7890             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7891             \l_@@_tmpd_dim
7892         }
7893         \CT@arc@
7894         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7895         \pgfsetrectcap
7896         \pgfusepathqstroke
7897       }
7898     }
7899   \@@_qpoint:n { row - #1 - base }

```



```

7900 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7901 \int_step_inline:nn { \l_@@_split_int }
7902 {
7903   \group_begin:
7904   \dim_set:Nn \col@sep
7905     { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7906   \pgftransformshift
7907     {
7908     \pgfpoint
7909       {
7910         \l_tmpa_dim + ##1 \l_tmpb_dim -
7911         \str_case:on \l_@@_hpos_block_str
7912           {
7913             l { \l_tmpb_dim + \col@sep}
7914             c { 0.5 \l_tmpb_dim }
7915             r { \col@sep }
7916           }
7917         }
7918       { \l_@@_tmpc_dim }
7919     }
7920   \pgfset { inner~sep = \c_zero_dim }
7921   \pgfnode
7922     { rectangle }
7923     {
7924       \str_case:on \l_@@_hpos_block_str
7925         {
7926           c { base }
7927           l { base-west }
7928           r { base-east }
7929         }
7930       }
7931     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7932   \group_end:
7933 }
7934 \endpgfpicture
7935 }

```

Now the case where there is no ampersand & in the content of the block.

```

7936 {
7937   \bool_if:NTF \l_@@_p_block_bool
7938   {

```

When the final user has used the key p, we have to compute the width.

```

7939 \pgfpicture
7940 \pgfrememberpicturepositiononpagetrue
7941 \pgf@relevantforpicturesizefalse
7942 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7943 {
7944   \@@_qpoint:n { col - #2 }
7945   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7946   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7947 }
7948 {
7949   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7950   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7951   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7952 }
7953 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7954 \endpgfpicture
7955 \hbox_set:Nn \l_@@_cell_box
7956 {
7957   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7958     { \g_tmpb_dim }
7959     \str_case:on \l_@@_hpos_block_str

```

```

7960         { c \centering r \raggedleft l \raggedright j { } }
7961         #6
7962         \end { minipage }
7963     }
7964 }
7965 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7966 \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.

```

7967 \pgfpicture
7968 \pgfrememberpicturepositiononpagetrue
7969 \pgf@relevantforpicturesizefalse
7970 \bool_lazy_any:nTF
7971 {
7972     { \str_if_empty_p:N \l_@@_vpos_block_str }
7973     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7974     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7975     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7976 }
7977 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7978     \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`.

```

7979     \bool_if:nT \g_@@_last_col_found_bool
7980     {
7981         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
7982         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7983     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7984     \tl_set:Ne \l_tmpa_tl
7985     {
7986         \str_case:on \l_@@_vpos_block_str
7987         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7988         { } {
7989             \str_case:on \l_@@_hpos_block_str
7990             {
7991                 c { center }
7992                 l { west }
7993                 r { east }
7994                 j { center }
7995             }
7996         }
7997     c {
7998         \str_case:on \l_@@_hpos_block_str
7999         {
8000             c { center }
8001             l { west }
8002             r { east }
8003             j { center }
8004         }
8005     }
8006 }
8007 T {
8008     \str_case:on \l_@@_hpos_block_str
8009     {
8010         c { north }

```

```

8011         l { north-west }
8012         r { north-east }
8013         j { north }
8014     }
8015
8016     }
8017     B {
8018         \str_case:on \l_@@_hpos_block_str
8019         {
8020             c { south }
8021             l { south-west }
8022             r { south-east }
8023             j { south }
8024         }
8025
8026     }
8027 }
8028
8029 \pgftransformshift
8030 {
8031     \pgfpointanchor
8032     {
8033         \@@_env: - #1 - #2 - block
8034         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8035     }
8036     { \l_tmpa_tl }
8037 }
8038 \pgfset { inner-sep = \c_zero_dim }
8039 \pgfnode
8040 { rectangle }
8041 { \l_tmpa_tl }
8042 { \box_use_drop:N \l_@@_cell_box } { } { }
8043 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

8044 {
8045     \pgfextracty \l_tmpa_dim
8046     {
8047         \@@_qpoint:n
8048         {
8049             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8050             - base
8051         }
8052     }
8053     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

8054     \pgfpointanchor
8055     {
8056         \@@_env: - #1 - #2 - block
8057         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8058     }
8059     {
8060         \str_case:on \l_@@_hpos_block_str
8061         {
8062             c { center }
8063             l { west }
8064             r { east }
8065             j { center }
8066         }
8067     }

```

We put the label of the block which has been composed in $\l_@@_cell_box$.

```

8068     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }

```

```

8069         \pgfset { inner~sep = \c_zero_dim }
8070         \pgfnode
8071         { rectangle }
8072         {
8073             \str_case:on \l_@@_hpos_block_str
8074             {
8075                 c { base }
8076                 l { base~west }
8077                 r { base~east }
8078                 j { base }
8079             }
8080         }
8081         { \box_use_drop:N \l_@@_cell_box } { } { }
8082     }
8083     \endpgfpicture
8084 }
8085 \group_end:
8086 }
8087 \cs_generate_variant:Nn \@@_Block_v:nnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8088 \cs_set_protected:Npn \@@_fill:nnnn #1 #2 #3 #4 #5
8089 {
8090     \pgfpicture
8091     \pgfrememberpicturepositiononpagetrue
8092     \pgf@relevantforpicturesizefalse
8093     \pgfpathrectanglecorners
8094     { \pgfpoint { #2 } { #3 } }
8095     { \pgfpoint { #4 } { #5 } }
8096     \pgfsetfillcolor { #1 }
8097     \pgfusepath { fill }
8098     \endpgfpicture
8099 }

```

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).

```

8100 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8101 {
8102     \tl_if_empty:NF \l_@@_opacity_tl
8103     {
8104         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8105             {
8106                 \tl_set:Ne \l_@@_fill_tl
8107                 {
8108                     [ opacity = \l_@@_opacity_tl ,
8109                     \tl_tail:o \l_@@_fill_tl
8110                 }
8111             }
8112             {
8113                 \tl_set:Ne \l_@@_fill_tl
8114                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8115             }
8116         }
8117     }

```

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).

```

8118 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3

```

```

8119 {
8120   \group_begin:
8121   \tl_clear:N \l_@@_draw_tl
8122   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8123   \keys_set_known:n { nicematrix / BlockStroke } { #1 }
8124   \pgfpicture
8125   \pgfrememberpicturepositiononpagetrue
8126   \pgf@relevantforpicturesizefalse
8127   \tl_if_empty:NF \l_@@_draw_tl
8128   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8129     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8130     { \CT@arc@ }
8131     { \@@_color:o \l_@@_draw_tl }
8132   }
8133   \pgfsetcornersarced
8134   {
8135     \pgfpoint
8136     { \l_@@_rounded_corners_dim }
8137     { \l_@@_rounded_corners_dim }
8138   }
8139   \@@_cut_on_hyphen:w #2 \q_stop
8140   \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8141   {
8142     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8143     {
8144       \@@_qpoint:n { row - \l_tmpa_tl }
8145       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8146       \@@_qpoint:n { col - \l_tmpb_tl }
8147       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8148       \@@_cut_on_hyphen:w #3 \q_stop
8149       \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8150       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8151       \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8152       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8153       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8154       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8155       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8156       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8157       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8158       \pgfpathrectanglecorners
8159       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8160       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8161       \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8162       { \pgfusepathqstroke }
8163       { \pgfusepath { stroke } }
8164     }
8165   }
8166   \endpgfpicture
8167   \group_end:
8168 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8169 \keys_define:nn { nicematrix / BlockStroke }
8170 {
8171   color .tl_set:N = \l_@@_draw_tl ,
8172   draw .code:n =
8173     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8174   draw .default:n = default ,
8175   line-width .dim_set:N = \l_@@_line_width_dim ,
8176   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8177   rounded-corners .default:n = 4 pt

```

8178 } }

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).

```
8179 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8180 {
8181   \group_begin:
8182   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8183   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8184   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8185   \@@_cut_on_hyphen:w #2 \q_stop
8186   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8187   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8188   \@@_cut_on_hyphen:w #3 \q_stop
8189   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8190   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8191   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8192   {
8193     \use:e
8194     {
8195       \@@_vline:n
8196       {
8197         position = ##1 ,
8198         start = \l_@@_tmpc_tl ,
8199         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8200         total-width = \dim_use:N \l_@@_line_width_dim
8201       }
8202     }
8203   }
8204   \group_end:
8205 }

8206 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8207 {
8208   \group_begin:
8209   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8210   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8211   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8212   \@@_cut_on_hyphen:w #2 \q_stop
8213   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8214   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8215   \@@_cut_on_hyphen:w #3 \q_stop
8216   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8217   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8218   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8219   {
8220     \use:e
8221     {
8222       \@@_hline:n
8223       {
8224         position = ##1 ,
8225         start = \l_@@_tmpd_tl ,
8226         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8227         total-width = \dim_use:N \l_@@_line_width_dim
8228       }
8229     }
8230   }
8231   \group_end:
8232 }
```

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).

```

8233 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8234 {
8235   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8236   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8237   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8238     { \@@_error:n { borders~forbidden } }
8239     {
8240       \tl_clear_new:N \l_@@_borders_tikz_tl
8241       \keys_set:no
8242         { nicematrix / OnlyForTikzInBorders }
8243         \l_@@_borders_clist
8244         \@@_cut_on_hyphen:w #2 \q_stop
8245         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8246         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8247         \@@_cut_on_hyphen:w #3 \q_stop
8248         \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8249         \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8250         \@@_stroke_borders_block_i:
8251       }
8252   }
8253 \hook_gput_code:nnn { begindocument } { . }
8254 {
8255   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8256     {
8257       \c_@@_pgfortikzpicture_tl
8258       \@@_stroke_borders_block_ii:
8259       \c_@@_endpgfortikzpicture_tl
8260     }
8261   }
8262 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8263 {
8264   \pgfrememberpicturepositiononpagetrue
8265   \pgf@relevantforpicturesizefalse
8266   \CT@arc@
8267   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8268   \clist_if_in:NnT \l_@@_borders_clist { right }
8269     { \@@_stroke_vertical:n \l_tmpb_tl }
8270   \clist_if_in:NnT \l_@@_borders_clist { left }
8271     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8272   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8273     { \@@_stroke_horizontal:n \l_tmpa_tl }
8274   \clist_if_in:NnT \l_@@_borders_clist { top }
8275     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8276   }
8277 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8278 {
8279   tikz .code:n =
8280     \cs_if_exist:NTF \tikzpicture
8281       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8282       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8283   tikz .value_required:n = true ,
8284   top .code:n = ,
8285   bottom .code:n = ,
8286   left .code:n = ,
8287   right .code:n = ,
8288   unknown .code:n = \@@_error:n { bad~border }
8289 }

```

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).

```

8290 \cs_new_protected:Npn \@@_stroke_vertical:n #1

```

```

8291 {
8292   \@@_qpoint:n \l_@@_tmpc_tl
8293   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8294   \@@_qpoint:n \l_tmpa_tl
8295   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8296   \@@_qpoint:n { #1 }
8297   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8298     {
8299       \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8300       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8301       \pgfusepathqstroke
8302     }
8303     {
8304       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8305         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8306     }
8307 }

```

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).

```

8308 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8309 {
8310   \@@_qpoint:n \l_@@_tmpd_tl
8311   \clist_if_in:NnTF \l_@@_borders_clist { left }
8312     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8313     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8314   \@@_qpoint:n \l_tmpb_tl
8315   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8316   \@@_qpoint:n { #1 }
8317   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8318     {
8319       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8320       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8321       \pgfusepathqstroke
8322     }
8323     {
8324       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8325         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8326     }
8327 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8328 \keys_define:nn { nicematrix / BlockBorders }
8329 {
8330   borders .clist_set:N = \l_@@_borders_clist ,
8331   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8332   rounded-corners .default:n = 4 pt ,
8333   line-width .dim_set:N = \l_@@_line_width_dim
8334 }

```

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.

```

8335 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8336 {
8337   \begin { tikzpicture }
8338   \@@_clip_with_rounded_corners:

```


We use `clist_map_inline:nn` because #5 is a list of lists.

```
8339 \clist_map_inline:nn { #1 }
8340 {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8341 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8342 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8343 (
8344 [
8345 xshift = \dim_use:N \l_@@_offset_dim ,
8346 yshift = - \dim_use:N \l_@@_offset_dim
8347 ]
8348 #2 -| #3
8349 )
8350 rectangle
8351 (
8352 [
8353 xshift = - \dim_use:N \l_@@_offset_dim ,
8354 yshift = \dim_use:N \l_@@_offset_dim
8355 ]
8356 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8357 ) ;
8358 }
8359 \end { tikzpicture }
8360 }
8361 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8362 \keys_define:nn { nicematrix / SpecialOffset }
8363 { 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.

```
8364 \cs_new_protected:Npn \@@_NullBlock:
8365 { \@@_collect_options:n { \@@_NullBlock_i: } }
8366 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8367 { }
```

27 How to draw the dotted lines transparently

```
8368 \cs_set_protected:Npn \@@_renew_matrix:
8369 {
8370 \RenewDocumentEnvironment { pmatrix } { }
8371 { \pNiceMatrix }
8372 { \endpNiceMatrix }
8373 \RenewDocumentEnvironment { vmatrix } { }
8374 { \vNiceMatrix }
8375 { \endvNiceMatrix }
8376 \RenewDocumentEnvironment { Vmatrix } { }
8377 { \VNiceMatrix }
8378 { \endVNiceMatrix }
8379 \RenewDocumentEnvironment { bmatrix } { }
8380 { \bNiceMatrix }
8381 { \endbNiceMatrix }
8382 \RenewDocumentEnvironment { Bmatrix } { }
8383 { \BNiceMatrix }
8384 { \endBNiceMatrix }
8385 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8386 \keys_define:nn { nicematrix / Auto }
8387 {
8388   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8389   columns-type .value_required:n = true ,
8390   l .meta:n = { columns-type = l } ,
8391   r .meta:n = { columns-type = r } ,
8392   c .meta:n = { columns-type = c } ,
8393   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8394   delimiters / color .value_required:n = true ,
8395   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8396   delimiters / max-width .default:n = true ,
8397   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8398   delimiters .value_required:n = true ,
8399   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8400   rounded-corners .default:n = 4 pt
8401 }

8402 \NewDocumentCommand \AutoNiceMatrixWithDelims
8403 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8404 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8405 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8406 {

```

The group is for the protection of the keys.

```

8407   \group_begin:
8408   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8409   \use:e
8410   {
8411     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8412     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8413     [ \exp_not:o \l_tmpa_tl ]
8414   }
8415   \int_if_zero:nT { \l_@@_first_row_int }
8416   {
8417     \int_if_zero:nT { \l_@@_first_col_int } { & }
8418     \prg_replicate:nn { #4 - 1 } { & }
8419     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8420   }
8421   \prg_replicate:nn { #3 }
8422   {
8423     \int_if_zero:nT { \l_@@_first_col_int } { & }

```

We put `{ }` before `#6` to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```

8424     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8425     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8426   }
8427   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8428   {
8429     \int_if_zero:nT { \l_@@_first_col_int } { & }
8430     \prg_replicate:nn { #4 - 1 } { & }
8431     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8432   }
8433   \end { NiceArrayWithDelims }
8434   \group_end:
8435 }

8436 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8437 {

```

```

8438 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8439 {
8440   \bool_gset_true:N \g_@@_delims_bool
8441   \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8442   \AutoNiceMatrixWithDelims { #2 } { #3 }
8443 }
8444 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8445 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8446 {
8447   \group_begin:
8448   \bool_gset_false:N \g_@@_delims_bool
8449   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8450   \group_end:
8451 }

```

29 The redefinition of the command `\dotfill`

```

8452 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8453 \cs_new_protected:Npn \@@_dotfill:
8454 {

```

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}`).

```

8455   \@@_old_dotfill:
8456   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8457 }

```

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`.

```

8458 \cs_new_protected:Npn \@@_dotfill_i:
8459 {
8460   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8461   { \@@_old_dotfill: }
8462 }

```

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.

```

8463 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8464 {
8465   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8466   {
8467     \@@_actually_diagbox:nnnnnn
8468     { \int_use:N \c@iRow }
8469     { \int_use:N \c@jCol }
8470     { \int_use:N \c@iRow }
8471     { \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.

```

8472     { \g_@@_row_style_tl \exp_not:n { #1 } }
8473     { \g_@@_row_style_tl \exp_not:n { #2 } }
8474 }

```

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.

```

8475 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8476 {
8477   { \int_use:N \c@iRow }
8478   { \int_use:N \c@jCol }
8479   { \int_use:N \c@iRow }
8480   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8481   { }
8482 }
8483 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnn` 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.

```

8484 \cs_new_protected:Npn \@@_actually_diagbox:nnnnn #1 #2 #3 #4 #5 #6
8485 {
8486   \pgfpicture
8487   \pgf@relevantforpicturesizefalse
8488   \pgfrememberpicturepositiononpagetrue
8489   \@@_qpoint:n { row - #1 }
8490   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8491   \@@_qpoint:n { col - #2 }
8492   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8493   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8494   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8495   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8496   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8497   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8498   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8499   {

```

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.

```

8500   \CT@arc@
8501   \pgfsetroundcap
8502   \pgfusepathqstroke
8503 }
8504 \pgfset { inner~sep = 1 pt }
8505 \pgfscope
8506 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8507 \pgfnode { rectangle } { south~west }
8508 {
8509   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8510   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8511   \end { minipage }
8512 }
8513 { }
8514 { }
8515 \endpgfscope
8516 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8517 \pgfnode { rectangle } { north~east }
8518 {
8519   \begin { minipage } { 20 cm }
8520   \raggedleft

```

```

8521     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8522     \end { minipage }
8523   }
8524   { }
8525   { }
8526 \endpgfpicture
8527 }

```

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`.

```

8528 \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 `\`.

```

8529 \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`.

```

8530 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8531 {
8532   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8533   \@@_CodeAfter_iv:n
8534 }

```

We catch the argument of the command `\end` (in `#1`).

```

8535 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8536 {

```

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.

```

8537   \str_if_eq:eeTF \@currenvir { #1 }
8538   { \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`.

```

8539   {
8540     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8541     \@@_CodeAfter_ii:n
8542   }
8543 }

```

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.

```

8544 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8545   {
8546     \pgfpicture
8547     \pgfrememberpicturepositiononpagetrue
8548     \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.

```

8549     \@@_qpoint:n { row - 1 }
8550     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8551     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8552     \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).

```

8553     \bool_if:nTF { #3 }
8554       { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8555       { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8556     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8557       {
8558         \cs_if_exist:cT
8559           { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8560           {
8561             \pgfpointanchor
8562               { \@@_env: - ##1 - #2 }
8563               { \bool_if:nTF { #3 } { west } { east } }
8564             \dim_set:Nn \l_tmpa_dim
8565               {
8566                 \bool_if:nTF { #3 }
8567                   { \dim_min:nn }
8568                   { \dim_max:nn }
8569                 \l_tmpa_dim
8570                 { \pgf@x }
8571               }
8572           }
8573       }

```

Now we can put the delimiter with a node of PGF.

```

8574     \pgfset { inner~sep = \c_zero_dim }
8575     \dim_zero:N \nulldelimiterspace
8576     \pgftransformshift
8577       {
8578         \pgfpoint
8579           { \l_tmpa_dim }
8580           { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8581       }
8582     \pgfnode
8583       { rectangle }
8584       { \bool_if:nTF { #3 } { east } { west } }
8585       {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8586     \nullfont
8587     \c_math_toggle_token
8588     \@@_color:o \l_@@_delimiters_color_tl
8589     \bool_if:nTF { #3 } { \left #1 } { \left . }
8590     \vcenter
8591       {
8592         \nullfont
8593         \hrule \@height

```

```

8594         \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8595         \@depth \c_zero_dim
8596         \@width \c_zero_dim
8597     }
8598     \bool_if:nTF { #3 } { \right . } { \right #1 }
8599     \c_math_toggle_token
8600 }
8601 { }
8602 { }
8603 \endpgfpicture
8604 }

```

33 The command `\SubMatrix`

```

8605 \keys_define:nn { nicematrix / sub-matrix }
8606 {
8607     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8608     extra-height .value_required:n = true ,
8609     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8610     left-xshift .value_required:n = true ,
8611     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8612     right-xshift .value_required:n = true ,
8613     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8614     xshift .value_required:n = true ,
8615     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8616     delimiters / color .value_required:n = true ,
8617     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8618     slim .default:n = true ,
8619     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8620     hlines .default:n = all ,
8621     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8622     vlines .default:n = all ,
8623     hvlines .meta:n = { hlines, vlines } ,
8624     hvlines .value_forbidden:n = true
8625 }
8626 \keys_define:nn { nicematrix }
8627 {
8628     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8629     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8630     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8631     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8632 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8633 \keys_define:nn { nicematrix / SubMatrix }
8634 {
8635     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8636     delimiters / color .value_required:n = true ,
8637     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8638     hlines .default:n = all ,
8639     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8640     vlines .default:n = all ,
8641     hvlines .meta:n = { hlines, vlines } ,
8642     hvlines .value_forbidden:n = true ,
8643     name .code:n =
8644     \tl_if_empty:nTF { #1 }
8645     { \@@_error:n { Invalid-name } }
8646     {
8647         \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }

```

```

8648     {
8649         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8650         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8651         {
8652             \str_set:Nn \l_@@_submatrix_name_str { #1 }
8653             \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8654         }
8655     }
8656     { \@@_error:n { Invalid~name } }
8657 },
8658 name .value_required:n = true ,
8659 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8660 rules .value_required:n = true ,
8661 code .tl_set:N = \l_@@_code_tl ,
8662 code .value_required:n = true ,
8663 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8664 }

8665 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8666 {
8667     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8668     {
8669         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8670         [
8671             delimiters / color = \l_@@_delimiters_color_tl ,
8672             hlines = \l_@@_submatrix_hlines_clist ,
8673             vlines = \l_@@_submatrix_vlines_clist ,
8674             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8675             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8676             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8677             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8678             #5
8679         ]
8680     }
8681     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8682     \ignorespaces
8683 }

8684 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8685 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8686 { \@@_SubMatrix_in_code_before_i:n n n n #1 #2 }

8687 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:n n n n #1 #2 #3 #4
8688 {
8689     \seq_gput_right:Ne \g_@@_submatrix_seq
8690     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8691     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8692     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8693     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8694     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8695 }
8696 }

```

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).

```

8697 \NewDocumentCommand \@@_compute_i_j:nn
8698 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8699 { \@@_compute_i_j:n n n n #1 #2 }

8700 \cs_new_protected:Npn \@@_compute_i_j:n n n n #1 #2 #3 #4
8701 {
8702     \def \l_@@_first_i_tl { #1 }

```



```

8703 \def \l_@@_first_j_tl { #2 }
8704 \def \l_@@_last_i_tl { #3 }
8705 \def \l_@@_last_j_tl { #4 }
8706 \tl_if_eq:NnT \l_@@_first_i_tl { last }
8707   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8708 \tl_if_eq:NnT \l_@@_first_j_tl { last }
8709   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8710 \tl_if_eq:NnT \l_@@_last_i_tl { last }
8711   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8712 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8713   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8714 }

```

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`.

```

8715 \hook_gput_code:nnn { begindocument } { . }
8716 {
8717   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8718   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8719     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8720 }
8721 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8722 {
8723   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8724 \@@_compute_i_j:nn { #2 } { #3 }
8725 \int_compare:nNtT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8726   { \def \arraystretch { 1 } }
8727 \bool_lazy_or:nnTF
8728   { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8729   { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8730   { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8731   {
8732     \str_clear_new:N \l_@@_submatrix_name_str
8733     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8734     \pgfpicture
8735     \pgfrememberpicturepositiononpagetrue
8736     \pgf@relevantforpicturesizefalse
8737     \pgfset { inner~sep = \c_zero_dim }
8738     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8739     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

8740 \bool_if:NTF \l_@@_submatrix_slim_bool
8741   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8742   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8743   {

```

```

8744     \cs_if_exist:cT
8745     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8746     {
8747         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8748         \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
8749         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8750     }
8751     \cs_if_exist:cT
8752     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8753     {
8754         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8755         \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
8756         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8757     }
8758 }
8759 \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
8760 { \@@_error:nn { Impossible~delimiter } { left } }
8761 {
8762     \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }
8763     { \@@_error:nn { Impossible~delimiter } { right } }
8764     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8765 }
8766 \endpgfpicture
8767 }
8768 \group_end:
8769 \ignorespaces
8770 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8771 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8772 {
8773     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8774     \dim_set:Nn \l_@@_y_initial_dim
8775     {
8776         \fp_to_dim:n
8777         {
8778             \pgf@y
8779             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8780         }
8781     }
8782     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8783     \dim_set:Nn \l_@@_y_final_dim
8784     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8785     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8786     {
8787         \cs_if_exist:cT
8788         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8789         {
8790             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8791             \dim_set:Nn \l_@@_y_initial_dim
8792             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8793         }
8794         \cs_if_exist:cT
8795         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8796         {
8797             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8798             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8799             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8800         }
8801     }
8802     \dim_set:Nn \l_tmpa_dim
8803     {
8804         \l_@@_y_initial_dim - \l_@@_y_final_dim +

```

```

8805     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8806   }
8807   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8808   \group_begin:
8809   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8810   \@@_set_CTarc:o \l_@@_rules_color_tl
8811   \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`.

```

8812   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8813   {
8814     \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8815     {
8816       \int_compare:nNnT
8817         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8818       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8819         \@@_qpoint:n { col - ##1 }
8820         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8821         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8822         \pgfusepathqstroke
8823       }
8824     }
8825   }

```

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.

```

8826   \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8827   { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8828   { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8829   {
8830     \bool_lazy_and:nnTF
8831     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8832     {
8833       \int_compare_p:nNn
8834       { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8835     {
8836       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8837       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8838       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8839       \pgfusepathqstroke
8840     }
8841     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8842   }

```

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.

```

8843   \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8844   { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8845   { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8846   {
8847     \bool_lazy_and:nnTF
8848     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8849     {
8850       \int_compare_p:nNn
8851       { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8852     {
8853       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8854     \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8855     \dim_set:Nn \l_tmpa_dim
8856     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8857     \str_case:nn { #1 }
8858     {
8859         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8860         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8861         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8862         }
8863     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8864     \dim_set:Nn \l_tmpb_dim
8865     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8866     \str_case:nn { #2 }
8867     {
8868         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8869         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8870         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8871     }
8872     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8873     \pgfusepathqstroke
8874     \group_end:
8875 }
8876 { \@@_error:mn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
8877 }

```

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).

```

8878     \str_if_empty:NF \l_@@_submatrix_name_str
8879     {
8880         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8881         \l_@@_x_initial_dim \l_@@_y_initial_dim
8882         \l_@@_x_final_dim \l_@@_y_final_dim
8883     }
8884     \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`.

```

8885     \begin { pgfscope }
8886     \pgftransformshift
8887     {
8888         \pgfpoint
8889         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8890         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8891     }
8892     \str_if_empty:NTF \l_@@_submatrix_name_str
8893     { \@@_node_left:nn #1 { } }
8894     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8895     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8896     \pgftransformshift
8897     {
8898         \pgfpoint
8899         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8900         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8901     }
8902     \str_if_empty:NTF \l_@@_submatrix_name_str

```

```

8903 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8904 {
8905   \@@_node_right:nnnn #2
8906   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8907 }

```

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`.

```

8908 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8909 \flag_clear_new:N \l_@@_code_flag
8910 \l_@@_code_tl
8911 }

```

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 $i-j$, $row-i$, $col-j$ 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`.

```

8912 \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:n` 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.

```

8913 \cs_new:Npn \@@_pgfpointanchor:n #1
8914 { \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`).

```

8915 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8916 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
8917 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8918 {

```

The command `\str_if_empty:nTF` is "fully expandable".

```

8919 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

8920 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

8921 { \@@_pgfpointanchor_ii:n { #1 } }
8922 }

```

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`).

```

8923 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8924 { \@@_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 or 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`.

```

8925 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

8926 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8927 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8928 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

8929 { \@@_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`).

```

8930 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8931 }

```

The following function is for the case when the name contains an hyphen.

```

8932 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8933 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8934 \@@_env:
8935 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8936 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8937 }

```

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.

```

8938 \tl_const:Nn \c_@@_integers_alist_tl
8939 {
8940 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8941 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8942 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8943 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8944 }

```

```

8945 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8946 {

```

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`.

```

8947 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8948 {
8949 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8950 \@@_env: -
8951 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8952 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8953 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8954 }
8955 {
8956 \str_if_eq:eeTF { #1 } { last }
8957 {
8958 \flag_raise:N \l_@@_code_flag
8959 \@@_env: -
8960 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8961 { \int_eval:n { \l_@@_last_i_tl + 1 } }
8962 { \int_eval:n { \l_@@_last_j_tl + 1 } }
8963 }

```

```

8964     { #1 }
8965   }
8966 }

```

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`).

```

8967 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8968 {
8969   \pgfnode
8970     { rectangle }
8971     { east }
8972     {
8973       \nullfont
8974       \c_math_toggle_token
8975       \@@_color:o \l_@@_delimiters_color_tl
8976       \left #1
8977       \vcenter
8978         {
8979           \nullfont
8980           \hrule \@height \l_tmpa_dim
8981             \depth \c_zero_dim
8982             \width \c_zero_dim
8983         }
8984       \right .
8985       \c_math_toggle_token
8986     }
8987     { #2 }
8988     { }
8989 }

```

The command `\@@_node_right:nnn` 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.

```

8990 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8991 {
8992   \pgfnode
8993     { rectangle }
8994     { west }
8995     {
8996       \nullfont
8997       \c_math_toggle_token
8998       \colorlet { current-color } { . }
8999       \@@_color:o \l_@@_delimiters_color_tl
9000       \left .
9001       \vcenter
9002         {
9003           \nullfont
9004           \hrule \@height \l_tmpa_dim
9005             \depth \c_zero_dim
9006             \width \c_zero_dim
9007         }
9008       \right #1
9009       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9010       ^ { \color { current-color } \smash { #4 } }
9011       \c_math_toggle_token
9012     }
9013     { #2 }
9014     { }
9015 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9016 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9017 {
9018   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9019   \ignorespaces
9020 }

9021 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9022 {
9023   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9024   \ignorespaces
9025 }

9026 \keys_define:nn { nicematrix / Brace }
9027 {
9028   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9029   left-shorten .default:n = true ,
9030   left-shorten .value_forbidden:n = true ,
9031   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9032   right-shorten .default:n = true ,
9033   right-shorten .value_forbidden:n = true ,
9034   shorten .meta:n = { left-shorten , right-shorten } ,
9035   shorten .value_forbidden:n = true ,
9036   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9037   yshift .value_required:n = true ,
9038   yshift .initial:n = \c_zero_dim ,
9039   color .tl_set:N = \l_tmpa_tl ,
9040   color .value_required:n = true ,
9041   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9042 }

```

#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`.

```

9043 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9044 {
9045   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9046 \@@_compute_i_j:nn { #1 } { #2 }
9047 \bool_lazy_or:nnTF
9048   { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9049   { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9050   {
9051     \str_if_eq:eeTF { #5 } { under }
9052     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9053     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9054   }
9055   {
9056     \tl_clear:N \l_tmpa_tl
9057     \keys_set:nn { nicematrix / Brace } { #4 }
9058     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9059     \pgfpicture
9060     \pgfrememberpicturepositiononpagetrue
9061     \pgf@relevantforpicturesizefalse
9062     \bool_if:NT \l_@@_brace_left_shorten_bool
9063     {
9064       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9065       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9066       {

```



```

9067     \cs_if_exist:cT
9068     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9069     {
9070     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9071
9072     \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9073     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9074     }
9075     }
9076   }
9077 \bool_lazy_or:nnT
9078 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9079 { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9080 {
9081   \@@_qpoint:n { col - \l_@@_first_j_tl }
9082   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9083 }
9084 \bool_if:NT \l_@@_brace_right_shorten_bool
9085 {
9086   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9087   \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9088   {
9089     \cs_if_exist:cT
9090     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9091     {
9092     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9093     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9094     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9095     }
9096   }
9097 }
9098 \bool_lazy_or:nnT
9099 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9100 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9101 {
9102   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9103   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9104 }
9105 \pgfset { inner~sep = \c_zero_dim }
9106 \str_if_eq:eeTF { #5 } { under }
9107 { \@@_underbrace_i:n { #3 } }
9108 { \@@_overbrace_i:n { #3 } }
9109 \endpgfpicture
9110 }
9111 \group_end:
9112 }

```

The argument is the text to put above the brace.

```

9113 \cs_new_protected:Npn \@@_overbrace_i:n #1
9114 {
9115   \@@_qpoint:n { row - \l_@@_first_i_tl }
9116   \pgftransformshift
9117   {
9118     \pgfpoint
9119     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9120     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9121   }
9122 \pgfnode
9123 { rectangle }
9124 { south }
9125 {
9126   \vtop
9127   {
9128     \group_begin:

```

```

9129     \everycr { }
9130     \halign
9131     {
9132         \hfil ## \hfil \crcr
9133         \bool_if:NTF \l_@@_tabular_bool
9134         { \begin { tabular } { c } #1 \end { tabular } }
9135         { $ \begin { array } { c } #1 \end { array } $ }
9136     \cr
9137     \c_math_toggle_token
9138     \overbrace
9139     {
9140         \hbox_to_wd:nn
9141         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9142         { }
9143     }
9144     \c_math_toggle_token
9145     \cr
9146     }
9147     \group_end:
9148 }
9149 }
9150 { }
9151 { }
9152 }

```

The argument is the text to put under the brace.

```

9153 \cs_new_protected:Npn \@@_underbrace_i:n #1
9154 {
9155     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9156     \pgftransformshift
9157     {
9158         \pgfpoint
9159         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9160         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9161     }
9162     \pgfnode
9163     { rectangle }
9164     { north }
9165     {
9166         \group_begin:
9167         \everycr { }
9168         \vbox
9169         {
9170             \halign
9171             {
9172                 \hfil ## \hfil \crcr
9173                 \c_math_toggle_token
9174                 \underbrace
9175                 {
9176                     \hbox_to_wd:nn
9177                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9178                     { }
9179                 }
9180                 \c_math_toggle_token
9181                 \cr
9182                 \bool_if:NTF \l_@@_tabular_bool
9183                 { \begin { tabular } { c } #1 \end { tabular } }
9184                 { $ \begin { array } { c } #1 \end { array } $ }
9185                 \cr
9186             }
9187         }
9188         \group_end:
9189     }
9190 }

```

```

9191     { }
9192 }

```

35 The commands HBrace et VBrace

```

9193 \hook_gput_code:nnn { begindocument } { . }
9194 {
9195   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9196   {
9197     \tikzset
9198     {
9199       nicematrix / brace / .style =
9200       {
9201         decoration = { brace , raise = -0.15 em } ,
9202         decorate ,
9203       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9204       nicematrix / mirrored-brace / .style =
9205       {
9206         nicematrix / brace ,
9207         decoration = mirror ,
9208       }
9209     }
9210   }
9211 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9212 \keys_define:nn { nicematrix / Hbrace }
9213 {
9214   color .code:n = ,
9215   horizontal-label .code:n = ,
9216   horizontal-labels .code:n = ,
9217   shorten .code:n = ,
9218   shorten-start .code:n = ,
9219   shorten-end .code:n = ,
9220   unknown .code:n = \@@_error:n { Unknown~key~for~Hbrace }
9221 }

```

Here we need an “fully expandable” command.

```

9222 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9223 {
9224   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9225   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9226   { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9227 }

```

The following command must *not* be protected.

```

9228 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9229 {
9230   \int_compare:nNnTF { \c@iRow } < { \c_one_int }
9231   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9232   \str_if_eq:nnTF { #2 } { * }
9233   {
9234     \NiceMatrixOptions { nullify-dots }

```

```

9235     \Ldots
9236     [
9237         line-style = nicematrix / brace ,
9238         #1 ,
9239         up =
9240         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9241     ]
9242 }
9243 {
9244     \Hdotsfor
9245     [
9246         line-style = nicematrix / brace ,
9247         #1 ,
9248         up =
9249         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9250     ]
9251     { #2 }
9252 }
9253 }
9254 {
9255     \str_if_eq:nnTF { #2 } { * }
9256     {
9257         \NiceMatrixOptions { nullify-dots }
9258         \Ldots
9259         [
9260             line-style = nicematrix / mirrored-brace ,
9261             #1 ,
9262             down =
9263             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9264         ]
9265     }
9266     {
9267         \Hdotsfor
9268         [
9269             line-style = nicematrix / mirrored-brace ,
9270             #1 ,
9271             down =
9272             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9273         ]
9274         { #2 }
9275     }
9276 }
9277 \keys_set:nn { nicematrix / Hbrace } { #1 }
9278 }

```

Here we need an “fully expandable” command.

```

9279 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9280 {
9281     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9282     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9283     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9284 }

```

The following command must *not* be protected.

```

9285 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9286 {
9287     \int_if_zero:nTF { \c@jCol }
9288     {
9289         \str_if_eq:nnTF { #2 } { * }
9290         {
9291             \NiceMatrixOptions { nullify-dots }
9292             \Vdots
9293             [
9294                 line-style = nicematrix / mirrored-brace ,

```

```

9295         #1 ,
9296         down =
9297         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9298     ]
9299 }
9300 {
9301     \Vdotsfor
9302     [
9303         line-style = nicematrix / mirrored-brace ,
9304         #1 ,
9305         down =
9306         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9307     ]
9308     { #2 }
9309 }
9310 }
9311 {
9312     \str_if_eq:nnTF { #2 } { * }
9313     {
9314         \NiceMatrixOptions { nullify-dots }
9315         \Vdots
9316         [
9317             line-style = nicematrix / brace ,
9318             #1 ,
9319             up =
9320             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9321         ]
9322     }
9323     {
9324         \Vdotsfor
9325         [
9326             line-style = nicematrix / brace ,
9327             #1 ,
9328             up =
9329             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9330         ]
9331         { #2 }
9332     }
9333 }
9334 \keys_set:nn { nicematrix / Hbrace } { #1 }
9335 }

```

36 The command TikzEveryCell

```

9336 \bool_new:N \l_@@_not_empty_bool
9337 \bool_new:N \l_@@_empty_bool
9338
9339 \keys_define:nn { nicematrix / TikzEveryCell }
9340 {
9341     not-empty .code:n =
9342         \bool_lazy_or:nnTF
9343         { \l_@@_in_code_after_bool }
9344         { \g_@@_create_cell_nodes_bool }
9345         { \bool_set_true:N \l_@@_not_empty_bool }
9346         { \@@_error:n { detection~of~empty~cells } } } ,
9347     not-empty .value_forbidden:n = true ,
9348     empty .code:n =
9349         \bool_lazy_or:nnTF
9350         { \l_@@_in_code_after_bool }
9351         { \g_@@_create_cell_nodes_bool }
9352         { \bool_set_true:N \l_@@_empty_bool }
9353         { \@@_error:n { detection~of~empty~cells } } } ,

```

```

9354     empty .value_forbidden:n = true ,
9355     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9356 }
9357
9358
9359 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9360 {
9361   \IfPackageLoadedTF { tikz }
9362   {
9363     \group_begin:
9364     \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.

```

9365     \tl_set:Nn \l_tmpa_tl { { #2 } }
9366     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9367     { \@@_for_a_block:nnnnn #1 }
9368     \@@_all_the_cells:
9369     \group_end:
9370   }
9371   { \@@_error:n { TikzEveryCell~without~tikz } }
9372 }
9373
9374 \tl_new:N \l_@@_i_tl
9375 \tl_new:N \l_@@_j_tl
9376
9377
9378 \cs_new_protected:Nn \@@_all_the_cells:
9379 {
9380   \int_step_inline:nn \c@iRow
9381   {
9382     \int_step_inline:nn \c@jCol
9383     {
9384       \cs_if_exist:cF { cell - ##1 - #####1 }
9385       {
9386         \clist_if_in:Nef \l_@@_corners_cells_clist
9387         { ##1 - #####1 }
9388         {
9389           \bool_set_false:N \l_tmpa_bool
9390           \cs_if_exist:cTF
9391           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9392           {
9393             \bool_if:NF \l_@@_empty_bool
9394             { \bool_set_true:N \l_tmpa_bool }
9395           }
9396           {
9397             \bool_if:NF \l_@@_not_empty_bool
9398             { \bool_set_true:N \l_tmpa_bool }
9399           }
9400           \bool_if:NT \l_tmpa_bool
9401           {
9402             \@@_block_tikz:nnnnn
9403             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9404           }
9405         }
9406       }
9407     }
9408   }
9409 }
9410
9411 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9412 {
9413   \bool_if:NF \l_@@_empty_bool
9414   {

```

```

9415     \@@_block_tikz:onn\nn
9416     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9417   }
9418   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9419 }
9420
9421 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9422 {
9423   \int_step_inline:nnn { #1 } { #3 }
9424   {
9425     \int_step_inline:nnn { #2 } { #4 }
9426     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9427   }
9428 }

```

37 The command `\ShowCellNames`

```

9429 \NewDocumentCommand \@@_ShowCellNames { }
9430 {
9431   \bool_if:NT \l_@@_in_code_after_bool
9432   {
9433     \pgfpicture
9434     \pgfrememberpicturepositiononpagetrue
9435     \pgf@relevantforpicturesizefalse
9436     \pgfpathrectanglecorners
9437     { \@@_qpoint:n { 1 } }
9438     {
9439       \@@_qpoint:n
9440       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9441     }
9442     \pgfsetfillopacity { 0.75 }
9443     \pgfsetfillcolor { white }
9444     \pgfusepathqfill
9445     \endpgfpicture
9446   }
9447   \dim_gzero_new:N \g_@@_tmpc_dim
9448   \dim_gzero_new:N \g_@@_tmpd_dim
9449   \dim_gzero_new:N \g_@@_tmpe_dim
9450   \int_step_inline:nn { \c@iRow }
9451   {
9452     \bool_if:NTF \l_@@_in_code_after_bool
9453     {
9454       \pgfpicture
9455       \pgfrememberpicturepositiononpagetrue
9456       \pgf@relevantforpicturesizefalse
9457     }
9458     { \begin { pgfpicture } }
9459     \@@_qpoint:n { row - ##1 }
9460     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9461     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9462     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9463     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9464     \bool_if:NTF \l_@@_in_code_after_bool
9465     { \endpgfpicture }
9466     { \end { pgfpicture } }
9467     \int_step_inline:nn { \c@jCol }
9468     {
9469       \hbox_set:Nn \l_tmpa_box
9470       {
9471         \normalfont \Large \sffamily \bfseries
9472         \bool_if:NTF \l_@@_in_code_after_bool
9473         { \color { red } }

```

```

9474         { \color { red ! 50 } }
9475     ##1 - ####1
9476     }
9477     \bool_if:NTF \l_@@_in_code_after_bool
9478     {
9479         \pgfpicture
9480         \pgfrememberpicturepositiononpagetrue
9481         \pgf@relevantforpicturesizefalse
9482     }
9483     { \begin { pgfpicture } }
9484     \@@_qpoint:n { col - ####1 }
9485     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9486     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9487     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9488     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9489     \bool_if:NTF \l_@@_in_code_after_bool
9490     { \endpgfpicture }
9491     { \end { pgfpicture } }
9492     \fp_set:Nn \l_tmpa_fp
9493     {
9494         \fp_min:nn
9495         {
9496             \fp_min:nn
9497             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9498             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9499         }
9500         { 1.0 }
9501     }
9502     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9503     \pgfpicture
9504     \pgfrememberpicturepositiononpagetrue
9505     \pgf@relevantforpicturesizefalse
9506     \pgftransformshift
9507     {
9508         \pgfpoint
9509         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9510         { \dim_use:N \g_tmpa_dim }
9511     }
9512     \pgfnode
9513     { rectangle }
9514     { center }
9515     { \box_use:N \l_tmpa_box }
9516     { }
9517     { }
9518     \endpgfpicture
9519     }
9520 }
9521 }

```

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.

```
9522 \bool_new:N \g_@@_footnotehyper_bool
```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quicky, it will also be set to `true` if the option `footnotehyper` is used.


```

9523 \bool_new:N \g_@@_footnote_bool
9524 \msg_new:nmmn { nicematrix } { Unknown~key~for~package }
9525 {
9526   You-have-used-the-key~' \l_keys_key_str '-when-loading-nicematrix-
9527   but-that-key-is-unknown. \\
9528   It-will-be-ignored. \\
9529   For-a-list-of-the-available-keys,~type-H~<return>.
9530 }
9531 {
9532   The-available-keys-are-(in-alphabetic-order):~
9533   footnote,~
9534   footnotehyper,~
9535   messages-for-Overleaf,~
9536   renew-dots~and~
9537   renew-matrix.
9538 }
9539 \keys_define:nn { nicematrix }
9540 {
9541   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9542   renew-dots .value_forbidden:n = true ,
9543   renew-matrix .code:n = \@@_renew_matrix: ,
9544   renew-matrix .value_forbidden:n = true ,
9545   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9546   footnote .bool_set:N = \g_@@_footnote_bool ,
9547   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9548   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9549 }
9550 \ProcessKeyOptions

9551 \@@_msg_new:nn { footnote-with-footnotehyper~package }
9552 {
9553   You-can't-use-the-option~'footnote'~because-the-package~
9554   footnotehyper-has-already-been-loaded.~
9555   If-you-want,~you-can-use-the-option~'footnotehyper'~and-the-footnotes~
9556   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9557   of-the-package~footnotehyper.\\
9558   The-package-footnote-won't-be-loaded.
9559 }
9560 \@@_msg_new:nn { footnotehyper-with-footnote~package }
9561 {
9562   You-can't-use-the-option~'footnotehyper'~because-the~package~
9563   footnote-has-already-been-loaded.~
9564   If-you-want,~you-can-use-the-option~'footnote'~and-the-footnotes~
9565   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9566   of-the-package~footnote.\\
9567   The-package-footnotehyper-won't-be-loaded.
9568 }

9569 \bool_if:NT \g_@@_footnote_bool
9570 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9571 \IfClassLoadedTF { beamer }
9572 { \bool_set_false:N \g_@@_footnote_bool }
9573 {
9574   \IfPackageLoadedTF { footnotehyper }
9575   { \@@_error:n { footnote-with-footnotehyper~package } }
9576   { \usepackage { footnote } }
9577 }
9578 }

```

```

9579 \bool_if:NT \g_@@_footnotehyper_bool
9580 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9581   \IfClassLoadedTF { beamer }
9582     { \bool_set_false:N \g_@@_footnote_bool }
9583     {
9584       \IfPackageLoadedTF { footnote }
9585         { \@@_error:n { footnotehyper~with~footnote~package } }
9586         { \usepackage { footnotehyper } }
9587     }
9588   \bool_set_true:N \g_@@_footnote_bool
9589 }

```

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.

```

9590 \bool_new:N \l_@@_underscore_loaded_bool
9591 \IfPackageLoadedT { underscore }
9592 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9593 \hook_gput_code:nnn { begindocument } { . }
9594 {
9595   \bool_if:NF \l_@@_underscore_loaded_bool
9596   {
9597     \IfPackageLoadedT { underscore }
9598       { \@@_error:n { underscore~after~nicematrix } }
9599   }
9600 }

```

40 Error messages of the package

```

9601 \str_const:Ne \c_@@_available_keys_str
9602 {
9603   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9604     { For~a~list~of~the~available~keys,~type~H~<return>. }
9605     { }
9606 }
9607 \seq_new:N \g_@@_types_of_matrix_seq
9608 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9609 {
9610   NiceMatrix ,
9611   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9612 }
9613 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9614 { \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`.

```

9615 \cs_new_protected:Npn \@@_error_too_much_cols:
9616 {
9617   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9618   { \@@_fatal:nn { too-much-cols-for-array } }
9619   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9620   { \@@_fatal:n { too-much-cols-for-matrix } }
9621   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9622   { \@@_fatal:n { too-much~cols~for~matrix } }
9623   \bool_if:NF \l_@@_last_col_without_value_bool
9624   { \@@_fatal:n { too-much~cols~for~matrix~with~last~col } }
9625 }

```

The following command must *not* be protected since it's used in an error message.

```

9626 \cs_new:Npn \@@_message_hdotsfor:
9627 {
9628   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9629   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ is~incorrect. }
9630 }
9631 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9632 {
9633   Incompatible~options.\\
9634   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9635   The~output~will~not~be~reliable.
9636 }
9637 \@@_msg_new:nn { key~color~inside }
9638 {
9639   Key~deprecated.\\
9640   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9641   and~have~been~deprecated.\\
9642   You~won't~have~similar~message~till~the~end~of~the~document.
9643 }
9644 \@@_msg_new:nn { invalid~weight }
9645 {
9646   Unknown~key.\\
9647   The~key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.
9648 }
9649 \@@_msg_new:nn { last~col~not~used }
9650 {
9651   Column~not~used.\\
9652   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9653   in~your~\@@_full_name_env: .~
9654   However,~you~can~go~on.
9655 }
9656 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9657 {
9658   Too~much~columns.\\
9659   In~the~row~ \int_eval:n { \c@iRow },~
9660   you~try~to~use~more~columns~
9661   than~allowed~by~your~ \@@_full_name_env: .
9662   \@@_message_hdotsfor: \
9663   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9664   (plus~the~exterior~columns).~This~error~is~fatal.
9665 }
9666 \@@_msg_new:nn { too~much~cols~for~matrix }
9667 {
9668   Too~much~columns.\\
9669   In~the~row~ \int_eval:n { \c@iRow } ,~
9670   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9671   \@@_message_hdotsfor: \
9672   Recall~that~the~maximal~number~of~columns~for~a~matrix~

```

```

9673 (excepted-the-potential-exterior-columns)-is-fixed-by-the-
9674 LaTeX-counter-'MaxMatrixCols'.~
9675 Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
9676 (use~ \token_to_str:N \setcounter \ to-change-that-value).~
9677 This-error-is-fatal.
9678 }

9679 \@@_msg_new:nn { too-much-cols-for-array }
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~
9684   \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
9685   \int_use:N \g_@@_static_num_of_col_int \
9686   \bool_if:nT
9687     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9688     { ~(plus-the-exterior-ones) }
9689   since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9690   This-error-is-fatal.
9691 }

9692 \@@_msg_new:nn { columns-not-used }
9693 {
9694   Columns-not-used.\\
9695   The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
9696   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
9697   columns-but-you-only-used~ \int_use:N \c@jCol .\\
9698   The-columns-you-did-not-used-won't-be-created.\\
9699   You-won't-have-similar-warning-till-the-end-of-the-document.
9700 }

9701 \@@_msg_new:nn { empty-preamble }
9702 {
9703   Empty-preamble.\\
9704   The-preamble-of-your~ \@@_full_name_env: \ is-empty.\\
9705   This-error-is-fatal.
9706 }

9707 \@@_msg_new:nn { in-first-col }
9708 {
9709   Erroneous-use.\\
9710   You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9711   That-command-will-be-ignored.
9712 }

9713 \@@_msg_new:nn { in-last-col }
9714 {
9715   Erroneous-use.\\
9716   You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9717   That-command-will-be-ignored.
9718 }

9719 \@@_msg_new:nn { in-first-row }
9720 {
9721   Erroneous-use.\\
9722   You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9723   That-command-will-be-ignored.
9724 }

9725 \@@_msg_new:nn { in-last-row }
9726 {
9727   Erroneous-use.\\
9728   You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9729   That-command-will-be-ignored.
9730 }

9731 \@@_msg_new:nn { TopRule-without-booktabs }
9732 {

```

```

9733     Erroneous~use.\\
9734     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9735     That~command~will~be~ignored.
9736   }

9737   \@@_msg_new:nn { TopRule~without~tikz }
9738   {
9739     Erroneous~use.\\
9740     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9741     That~command~will~be~ignored.
9742   }

9743   \@@_msg_new:nn { caption~outside~float }
9744   {
9745     Key~caption~forbidden.\\
9746     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9747     environment~(such~as~\{table\}).~This~key~will~be~ignored.
9748   }

9749   \@@_msg_new:nn { short~caption~without~caption }
9750   {
9751     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9752     However,~your~'short~caption'~will~be~used~as~'caption'.
9753   }

9754   \@@_msg_new:nn { double~closing~delimiter }
9755   {
9756     Double~delimiter.\\
9757     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9758     delimiter.~This~delimiter~will~be~ignored.
9759   }

9760   \@@_msg_new:nn { delimiter~after~opening }
9761   {
9762     Double~delimiter.\\
9763     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9764     delimiter.~That~delimiter~will~be~ignored.
9765   }

9766   \@@_msg_new:nn { bad~option~for~line~style }
9767   {
9768     Bad~line~style.\\
9769     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9770     is~'standard'.~That~key~will~be~ignored.
9771   }

9772   \@@_msg_new:nn { corners~with~no~cell~nodes }
9773   {
9774     Incompatible~keys.\\
9775     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9776     is~in~force.\\
9777     If~you~go~on,~that~key~will~be~ignored.
9778   }

9779   \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9780   {
9781     Incompatible~keys.\\
9782     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9783     is~in~force.\\
9784     If~you~go~on,~those~extra~nodes~won't~be~created.
9785   }

9786   \@@_msg_new:nn { Identical~notes~in~caption }
9787   {
9788     Identical~tabular~notes.\\
9789     You~can't~put~several~notes~with~the~same~content~in~
9790     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9791     If~you~go~on,~the~output~will~probably~be~erroneous.
9792   }

```

```

9793 \@@_msg_new:nn { tabularnote~below~the~tabular }
9794 {
9795   \token_to_str:N \tabularnote \ forbidden\\
9796   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9797   of~your~tabular~because~the~caption~will~be~composed~below~
9798   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9799   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions'\\.\\
9800   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9801   no~similar~error~will~raised~in~this~document.
9802 }

9803 \@@_msg_new:nn { Unknown~key~for~rules }
9804 {
9805   Unknown~key\\.\\
9806   There~is~only~two~keys~available~here:~width~and~color\\.\\
9807   Your~key~' \l_keys_key_str '~will~be~ignored.
9808 }

9809 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9810 {
9811   Unknown~key\\.\\
9812   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9813   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9814   and~ \token_to_str:N \Vbrace \ are:~'color',~
9815   'horizontal~label(s)',~'shorten'~'shorten~end'~
9816   and~'shorten~start'.
9817 }

9818 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9819 {
9820   Unknown~key\\.\\
9821   There~is~only~two~keys~available~here:~
9822   'empty'~and~'not~empty'\\.\\
9823   Your~key~' \l_keys_key_str '~will~be~ignored.
9824 }

9825 \@@_msg_new:nn { Unknown~key~for~rotate }
9826 {
9827   Unknown~key\\.\\
9828   The~only~key~available~here~is~'c'\\.\\
9829   Your~key~' \l_keys_key_str '~will~be~ignored.
9830 }

9831 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9832 {
9833   Unknown~key\\.\\
9834   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9835   It~you~go~on,~you~will~probably~have~other~errors. \\
9836   \c_@@_available_keys_str
9837 }
9838 {
9839   The~available~keys~are~(in~alphabetic~order):~
9840   ccommand,~
9841   color,~
9842   command,~
9843   dotted,~
9844   letter,~
9845   multiplicity,~
9846   sep~color,~
9847   tikz,~and~total~width.
9848 }

9849 \@@_msg_new:nnn { Unknown~key~for~xdots }
9850 {
9851   Unknown~key\\.\\
9852   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules\\.\\
9853   \c_@@_available_keys_str

```

```

9854 }
9855 {
9856   The-available-keys-are-(in-alphabetic-order):~
9857   'color',~
9858   'horizontal(s)-labels',~
9859   'inter',~
9860   'line-style',~
9861   'radius',~
9862   'shorten',~
9863   'shorten-end'~and~'shorten-start'.
9864 }

9865 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9866 {
9867   Unknown-key.\\
9868   As-for-now,~there-is~only~two-keys~available~here:~'cols'~and~'respect-blocks'~
9869   (and-you-try-to-use~'\l_keys_key_str '~)\\
9870   That-key-will-be-ignored.
9871 }

9872 \@@_msg_new:nn { label-without-caption }
9873 {
9874   You-can't-use-the-key~'label'~in-your~\{NiceTabular\}~because~
9875   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9876 }

9877 \@@_msg_new:nn { W-warning }
9878 {
9879   Line~ \msg_line_number: .~The-cell-is-too-wide-for-your-column~'W'~
9880   (row~ \int_use:N \c@iRow ).
9881 }

9882 \@@_msg_new:nn { Construct-too-large }
9883 {
9884   Construct-too-large.\\
9885   Your-command~ \token_to_str:N #1
9886   can't-be-drawn-because-your-matrix-is-too-small.\\
9887   That-command-will-be-ignored.
9888 }

9889 \@@_msg_new:nn { underscore-after-nicematrix }
9890 {
9891   Problem-with~'underscore'.\\
9892   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9893   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9894   '\token_to_str:N \Cdots \token_to_str:N _
9895   \{ n \token_to_str:N \text \{ ~times \} \}'.
9896 }

9897 \@@_msg_new:nn { ampersand-in-light-syntax }
9898 {
9899   Ampersand-forbidden.\\
9900   You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns-because~
9901   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9902 }

9903 \@@_msg_new:nn { double-backslash-in-light-syntax }
9904 {
9905   Double-backslash-forbidden.\\
9906   You-can't-use~ \token_to_str:N \\
9907   ~to-separate-rows-because-the-key~'light-syntax'~
9908   is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl '~
9909   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9910 }

9911 \@@_msg_new:nn { hlines-with-color }
9912 {
9913   Incompatible-keys.\\

```

```

9914     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9915     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
9916     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
9917     Your~key~will~be~discarded.
9918   }
9919   \@@_msg_new:nn { bad~value~for~baseline }
9920   {
9921     Bad~value~for~baseline.\\
9922     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
9923     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9924     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
9925     the~form~'line-i'.\\
9926     A~value~of~1~will~be~used.
9927   }
9928   \@@_msg_new:nn { detection~of~empty~cells }
9929   {
9930     Problem~with~'not~empty'\\
9931     For~technical~reasons,~you~must~activate~
9932     'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
9933     in~order~to~use~the~key~' \l_keys_key_str '.\\
9934     That~key~will~be~ignored.
9935   }
9936   \@@_msg_new:nn { siunitx~not~loaded }
9937   {
9938     siunitx~not~loaded\\
9939     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9940     That~error~is~fatal.
9941   }
9942   \@@_msg_new:nn { Invalid~name }
9943   {
9944     Invalid~name.\\
9945     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
9946     \SubMatrix \ of~your~ \@@_full_name_env: .\\
9947     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9948     This~key~will~be~ignored.
9949   }
9950   \@@_msg_new:nn { Hbrace~not~allowed }
9951   {
9952     Command~not~allowed.\\
9953     You~can't~use~the~command~ \token_to_str:N #1
9954     because~you~have~not~loaded~
9955     \IfPackageLoadedTF { tikz }
9956     { the~TikZ~library~'decorations.pathreplacing'~Use~ }
9957     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9958     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9959     That~command~will~be~ignored.
9960   }
9961   \@@_msg_new:nn { Vbrace~not~allowed }
9962   {
9963     Command~not~allowed.\\
9964     You~can't~use~the~command~ \token_to_str:N \Vbrace \
9965     because~you~have~not~loaded~TikZ~
9966     and~the~TikZ~library~'decorations.pathreplacing'.\\
9967     Use: ~\token_to_str:N \usepackage \{tikz\}~
9968     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9969     That~command~will~be~ignored.
9970   }
9971   \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9972   {
9973     Wrong~line.\\
9974     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~

```



```

9975 \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
9976 number~is~not~valid.~It~will~be~ignored.
9977 }
9978 \@@_msg_new:nn { Impossible~delimiter }
9979 {
9980 Impossible~delimiter.\\
9981 It's~impossible~to~draw~the~#1~delimiter~of~your~
9982 \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
9983 in~that~column.
9984 \bool_if:NT \l_@@_submatrix_slim_bool
9985 { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9986 This~ \token_to_str:N \SubMatrix \ will~be~ignored.
9987 }
9988 \@@_msg_new:nnn { width~without~X~columns }
9989 {
9990 You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
9991 the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
9992 That~key~will~be~ignored.
9993 }
9994 {
9995 This~message~is~the~message~'width~without~X~columns'~
9996 of~the~module~'nicematrix'.~
9997 The~experimented~users~can~disable~that~message~with~
9998 \token_to_str:N \msg_redirect_name:nnn .\\
9999 }
10000
10001 \@@_msg_new:nn { key~multiplicity~with~dotted }
10002 {
10003 Incompatible~keys. \\
10004 You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10005 in~a~'custom~line'.~They~are~incompatible. \\
10006 The~key~'multiplicity'~will~be~discarded.
10007 }
10008 \@@_msg_new:nn { empty~environment }
10009 {
10010 Empty~environment.\\
10011 Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10012 }
10013 \@@_msg_new:nn { No~letter~and~no~command }
10014 {
10015 Erroneous~use.\\
10016 Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10017 key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10018 '~'command'~(to~draw~horizontal~rules).\\
10019 However,~you~can~go~on.
10020 }
10021 \@@_msg_new:nn { Forbidden~letter }
10022 {
10023 Forbidden~letter.\\
10024 You~can't~use~the~letter~'#1'~for~a~customized~line.~
10025 It~will~be~ignored.\\
10026 The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10027 }
10028 \@@_msg_new:nn { Several~letters }
10029 {
10030 Wrong~name.\\
10031 You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10032 have~used~' \l_@@_letter_str ').\\
10033 It~will~be~ignored.
10034 }

```

```

10035 \@@_msg_new:nn { Delimiter-with-small }
10036 {
10037   Delimiter~forbidden.\\
10038   You~can't~put~a~delimiter~in~the~preamble~of~your~
10039   \@@_full_name_env: \
10040   because~the~key~'small'~is~in~force.\\
10041   This~error~is~fatal.
10042 }
10043 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10044 {
10045   Unknown~cell.\\
10046   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10047   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10048   can't~be~executed~because~a~cell~doesn't~exist.\\
10049   This~command~ \token_to_str:N \line \ will~be~ignored.
10050 }
10051 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10052 {
10053   Duplicate~name.\\
10054   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10055   in~this~ \@@_full_name_env: .\\
10056   This~key~will~be~ignored.\\
10057   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10058   { For~a~list~of~the~names~already~used,~type-H~<return>. }
10059 }
10060 {
10061   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10062   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10063 }
10064 \@@_msg_new:nn { r~or~l~with~preamble }
10065 {
10066   Erroneous~use.\\
10067   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10068   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10069   your~ \@@_full_name_env: .\\
10070   This~key~will~be~ignored.
10071 }
10072 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10073 {
10074   Erroneous~use.\\
10075   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10076   the~array.~This~error~is~fatal.
10077 }
10078 \@@_msg_new:nn { bad~corner }
10079 {
10080   Bad~corner.\\
10081   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10082   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10083   This~specification~of~corner~will~be~ignored.
10084 }
10085 \@@_msg_new:nn { bad~border }
10086 {
10087   Bad~border.\\
10088   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10089   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10090   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10091   also~use~the~key~'tikz'
10092   \IfPackageLoadedF { tikz }
10093   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10094   This~specification~of~border~will~be~ignored.
10095 }

```

```

10096 \@@_msg_new:nn { TikzEveryCell-without-tikz }
10097 {
10098   TikZ-not-loaded.\\
10099   You-can't-use \token_to_str:N \TikzEveryCell \
10100   because-you-have-not-loaded-tikz.~
10101   This-command-will-be-ignored.
10102 }
10103 \@@_msg_new:nn { tikz-key-without-tikz }
10104 {
10105   TikZ-not-loaded.\\
10106   You-can't-use-the-key~'tikz'~for-the-command~' \token_to_str:N
10107   \Block '~because-you-have-not-loaded-tikz.~
10108   This-key-will-be-ignored.
10109 }
10110 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10111 {
10112   Erroneous-use.\\
10113   In-the~ \@@_full_name_env: ,~you-must-use-the-key~
10114   'last-col'~without-value.\\
10115   However,~you-can-go-on-for-this-time~
10116   (the-value~' \l_keys_value_tl '~will-be-ignored).
10117 }
10118 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10119 {
10120   Erroneous-use. \\
10121   In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10122   'last-col'~without-value. \\
10123   However,~you-can-go-on-for-this-time~
10124   (the-value~' \l_keys_value_tl '~will-be-ignored).
10125 }
10126 \@@_msg_new:nn { Block-too-large-1 }
10127 {
10128   Block-too-large. \\
10129   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is-
10130   too-small-for-that-block. \\
10131   This-block-and-maybe-others-will-be-ignored.
10132 }
10133 \@@_msg_new:nn { Block-too-large-2 }
10134 {
10135   Block-too-large. \\
10136   The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10137   \g_@@_static_num_of_col_int \
10138   columns-but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
10139   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10140   (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10141   This-block-and-maybe-others-will-be-ignored.
10142 }
10143 \@@_msg_new:nn { unknown-column-type }
10144 {
10145   Bad-column-type. \\
10146   The-column-type~'#1'~in-your~ \@@_full_name_env: \
10147   is-unknown. \\
10148   This-error-is-fatal.
10149 }
10150 \@@_msg_new:nn { unknown-column-type-S }
10151 {
10152   Bad-column-type. \\
10153   The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown. \\
10154   If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should-
10155   load-that-package. \\
10156   This-error-is-fatal.

```

```

10157 }
10158 \@@_msg_new:nn { tabularnote~forbidden }
10159 {
10160   Forbidden~command. \\
10161   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10162   ~here.~This~command~is~available~only~in~
10163   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10164   the~argument~of~a~command~\token_to_str:N \caption \ included~
10165   in~an~environment~\{table\}. \\
10166   This~command~will~be~ignored.
10167 }
10168 \@@_msg_new:nn { borders~forbidden }
10169 {
10170   Forbidden~key.\\
10171   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10172   because~the~option~'rounded~corners'~
10173   is~in~force~with~a~non~zero~value.\\
10174   This~key~will~be~ignored.
10175 }
10176 \@@_msg_new:nn { bottomrule~without~booktabs }
10177 {
10178   booktabs~not~loaded.\\
10179   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10180   loaded~'booktabs'.\\
10181   This~key~will~be~ignored.
10182 }
10183 \@@_msg_new:nn { enumitem~not~loaded }
10184 {
10185   enumitem~not~loaded. \\
10186   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10187   ~because~you~haven't~loaded~'enumitem'. \\
10188   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10189   ignored~in~the~document.
10190 }
10191 \@@_msg_new:nn { tikz~without~tikz }
10192 {
10193   Tikz~not~loaded. \\
10194   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10195   loaded.~If~you~go~on,~that~key~will~be~ignored.
10196 }
10197 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10198 {
10199   Tikz~not~loaded. \\
10200   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10201   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10202   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10203   use~that~custom~line.
10204 }
10205 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10206 {
10207   Tikz~not~loaded. \\
10208   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10209   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10210   That~key~will~be~ignored.
10211 }
10212 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10213 {
10214   Erroneous~use.\\
10215   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10216   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~

```

```

10217     The-key~ 'color'~will-be-discarded.
10218 }
10219 \@@_msg_new:nn { Wrong-last-row }
10220 {
10221     Wrong-number.\
10222     You-have-used~'last-row= \int_use:N \l_@@_last_row_int '~but-your~
10223     \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10224     If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
10225     last-row.~You-can-avoid~this-problem-by-using~'last-row'~
10226     without-value~(more~compilations~might~be-necessary).
10227 }
10228 \@@_msg_new:nn { Yet-in-env }
10229 {
10230     Nested-environments.\
10231     Environments-of~nicematrix~can't-be-nested.\
10232     This-error-is-fatal.
10233 }
10234 \@@_msg_new:nn { Outside-math-mode }
10235 {
10236     Outside-math-mode.\
10237     The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
10238     (and-not-in~ \token_to_str:N \vcenter ).\
10239     This-error-is-fatal.
10240 }
10241 \@@_msg_new:nn { One-letter-allowed }
10242 {
10243     Bad-name.\
10244     The-value~of~key~' \l_keys_key_str '~must-be~of~length~1~and~
10245     you-have-used~' \l_keys_value_tl '~.\
10246     It-will-be-ignored.
10247 }
10248 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10249 {
10250     Environment~\{TabularNote\}~forbidden.\
10251     You-must-use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10252     but~*before*~the~ \token_to_str:N \CodeAfter . \
10253     This-environment~\{TabularNote\}~will-be-ignored.
10254 }
10255 \@@_msg_new:nn { varwidth-not-loaded }
10256 {
10257     varwidth-not-loaded.\
10258     You-can't-use-the-column~type~'V'~because~'varwidth'~is-not~
10259     loaded.\
10260     Your~column~will~behave~like~'p'.
10261 }
10262 \@@_msg_new:nnn { Unknown-key-for-RulesBis }
10263 {
10264     Unknown-key.\
10265     Your~key~' \l_keys_key_str '~is-unknown~for~a~rule.\
10266     \c_@@_available_keys_str
10267 }
10268 {
10269     The-available-keys-are~(in~alphabetic~order):~
10270     color,~
10271     dotted,~
10272     multiplicity,~
10273     sep-color,~
10274     tikz,~and~total-width.
10275 }
10276

```

```

10277 \@@_msg_new:nnn { Unknown~key~for~Block }
10278 {
10279   Unknown~key. \\
10280   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10281   \token_to_str:N \Block . \\
10282   It~will~be~ignored. \\
10283   \c_@@_available_keys_str
10284 }
10285 {
10286   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10287   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10288   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10289   and~vlines.
10290 }
10291 \@@_msg_new:nnn { Unknown~key~for~Brace }
10292 {
10293   Unknown~key. \\
10294   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10295   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10296   It~will~be~ignored. \\
10297   \c_@@_available_keys_str
10298 }
10299 {
10300   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10301   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10302   right~shorten)~and~yshift.
10303 }
10304 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10305 {
10306   Unknown~key. \\
10307   The~key~' \l_keys_key_str '~is~unknown. \\
10308   It~will~be~ignored. \\
10309   \c_@@_available_keys_str
10310 }
10311 {
10312   The~available~keys~are~(in~alphabetic~order):~
10313   delimiters/color,~
10314   rules~(with~the~subkeys~'color'~and~'width'),~
10315   sub~matrix~(several~subkeys)~
10316   and~xdots~(several~subkeys).~
10317   The~latter~is~for~the~command~ \token_to_str:N \line .
10318 }
10319 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10320 {
10321   Unknown~key. \\
10322   The~key~' \l_keys_key_str '~is~unknown. \\
10323   It~will~be~ignored. \\
10324   \c_@@_available_keys_str
10325 }
10326 {
10327   The~available~keys~are~(in~alphabetic~order):~
10328   create~cell~nodes,~
10329   delimiters/color~and~
10330   sub~matrix~(several~subkeys).
10331 }
10332 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10333 {
10334   Unknown~key. \\
10335   The~key~' \l_keys_key_str '~is~unknown. \\
10336   That~key~will~be~ignored. \\
10337   \c_@@_available_keys_str
10338 }

```

```

10339 {
10340   The-available-keys-are-(in~alphabetic-order):~
10341   'delimiters/color',~
10342   'extra-height',~
10343   'hlines',~
10344   'hvlines',~
10345   'left-xshift',~
10346   'name',~
10347   'right-xshift',~
10348   'rules'~(with~the~subkeys~'color'~and~'width'),~
10349   'slim',~
10350   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10351   and~'right-xshift').\\
10352 }
10353 \@@_msg_new:nnn { Unknown~key~for~notes }
10354 {
10355   Unknown~key.\\
10356   The~key~' \l_keys_key_str '-is~unknown.\\
10357   That~key~will~be~ignored. \\
10358   \c_@@_available_keys_str
10359 }
10360 {
10361   The-available-keys-are-(in~alphabetic-order):~
10362   bottomrule,~
10363   code-after,~
10364   code-before,~
10365   detect-duplicates,~
10366   enumitem-keys,~
10367   enumitem-keys-para,~
10368   para,~
10369   label-in-list,~
10370   label-in-tabular~and~
10371   style.
10372 }
10373 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10374 {
10375   Unknown~key.\\
10376   The~key~' \l_keys_key_str '-is~unknown~for~the~command~
10377   \token_to_str:N \RowStyle . \\
10378   That~key~will~be~ignored. \\
10379   \c_@@_available_keys_str
10380 }
10381 {
10382   The-available-keys-are-(in~alphabetic-order):~
10383   bold,~
10384   cell-space-top-limit,~
10385   cell-space-bottom-limit,~
10386   cell-space-limits,~
10387   color,~
10388   fill~(alias:~rowcolor),~
10389   nb-rows,~
10390   opacity~and~
10391   rounded-corners.
10392 }
10393 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10394 {
10395   Unknown~key.\\
10396   The~key~' \l_keys_key_str '-is~unknown~for~the~command~
10397   \token_to_str:N \NiceMatrixOptions . \\
10398   That~key~will~be~ignored. \\
10399   \c_@@_available_keys_str
10400 }
10401 {

```

```

10402 The~available~keys~are~(in~alphabetic~order):~
10403 &~in~blocks,~
10404 allow~duplicate~names,~
10405 ampersand~in~blocks,~
10406 caption~above,~
10407 cell~space~bottom~limit,~
10408 cell~space~limits,~
10409 cell~space~top~limit,~
10410 code~for~first~col,~
10411 code~for~first~row,~
10412 code~for~last~col,~
10413 code~for~last~row,~
10414 corners,~
10415 custom~key,~
10416 create~extra~nodes,~
10417 create~medium~nodes,~
10418 create~large~nodes,~
10419 custom~line,~
10420 delimiters~(several~subkeys),~
10421 end~of~row,~
10422 first~col,~
10423 first~row,~
10424 hlines,~
10425 hvlines,~
10426 hvlines~except~borders,~
10427 last~col,~
10428 last~row,~
10429 left~margin,~
10430 light~syntax,~
10431 light~syntax~expanded,~
10432 matrix/columns~type,~
10433 no~cell~nodes,~
10434 notes~(several~subkeys),~
10435 nullify~dots,~
10436 pgf~node~code,~
10437 renew~dots,~
10438 renew~matrix,~
10439 respect~arraystretch,~
10440 rounded~corners,~
10441 right~margin,~
10442 rules~(with~the~subkeys~'color'~and~'width'),~
10443 small,~
10444 sub~matrix~(several~subkeys),~
10445 vlines,~
10446 xdots~(several~subkeys).
10447 }

```

For `{NiceArray}`, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

10448 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10449 {
10450   Unknown~key.\\
10451   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10452   \{NiceArray\}. \\
10453   That~key~will~be~ignored. \\
10454   \c_@@_available_keys_str
10455 }
10456 {
10457   The~available~keys~are~(in~alphabetic~order):~
10458   &~in~blocks,~
10459   ampersand~in~blocks,~
10460   b,~
10461   baseline,~
10462   c,~

```



```

10463 cell-space-bottom-limit,~
10464 cell-space-limits,~
10465 cell-space-top-limit,~
10466 code-after,~
10467 code-for-first-col,~
10468 code-for-first-row,~
10469 code-for-last-col,~
10470 code-for-last-row,~
10471 columns-width,~
10472 corners,~
10473 create-extra-nodes,~
10474 create-medium-nodes,~
10475 create-large-nodes,~
10476 extra-left-margin,~
10477 extra-right-margin,~
10478 first-col,~
10479 first-row,~
10480 hlines,~
10481 hvlines,~
10482 hvlines-except-borders,~
10483 last-col,~
10484 last-row,~
10485 left-margin,~
10486 light-syntax,~
10487 light-syntax-expanded,~
10488 name,~
10489 no-cell-nodes,~
10490 nullify-dots,~
10491 pgf-node-code,~
10492 renew-dots,~
10493 respect-arraystretch,~
10494 right-margin,~
10495 rounded-corners,~
10496 rules~(with~the~subkeys~'color'~and~'width'),~
10497 small,~
10498 t,~
10499 vlines,~
10500 xdots/color,~
10501 xdots/shorten-start,~
10502 xdots/shorten-end,~
10503 xdots/shorten-and~
10504 xdots/line-style.
10505 }

```

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`).

```

10506 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10507 {
10508   Unknown~key.\\
10509   The~key~' \l_keys_key_str '~is~unknown~for~the~
10510   \@@_full_name_env: . \\
10511   That~key~will~be~ignored. \\
10512   \c_@@_available_keys_str
10513 }
10514 {
10515   The~available~keys~are~(in~alphabetic~order):~
10516   &~in~blocks,~
10517   ampersand~in~blocks,~
10518   b,~
10519   baseline,~
10520   c,~
10521   cell-space-bottom-limit,~
10522   cell-space-limits,~
10523   cell-space-top-limit,~

```

```

10524 code-after,~
10525 code-for-first-col,~
10526 code-for-first-row,~
10527 code-for-last-col,~
10528 code-for-last-row,~
10529 columns-type,~
10530 columns-width,~
10531 corners,~
10532 create-extra-nodes,~
10533 create-medium-nodes,~
10534 create-large-nodes,~
10535 extra-left-margin,~
10536 extra-right-margin,~
10537 first-col,~
10538 first-row,~
10539 hlines,~
10540 hvlines,~
10541 hvlines-except-borders,~
10542 l,~
10543 last-col,~
10544 last-row,~
10545 left-margin,~
10546 light-syntax,~
10547 light-syntax-expanded,~
10548 name,~
10549 no-cell-nodes,~
10550 nullify-dots,~
10551 pgf-node-code,~
10552 r,~
10553 renew-dots,~
10554 respect-arraystretch,~
10555 right-margin,~
10556 rounded-corners,~
10557 rules~(with~the~subkeys~'color'~and~'width'),~
10558 small,~
10559 t,~
10560 vlines,~
10561 xdots/color,~
10562 xdots/shorten-start,~
10563 xdots/shorten-end,~
10564 xdots/shorten-and~
10565 xdots/line-style.
10566 }
10567 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10568 {
10569   Unknown~key.\\
10570   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10571   \{NiceTabular\}. \\
10572   That~key~will~be~ignored. \\
10573   \c_@@_available_keys_str
10574 }
10575 {
10576   The~available~keys~are~(in~alphabetic~order):~
10577   &~in~blocks,~
10578   ampersand~in~blocks,~
10579   b,~
10580   baseline,~
10581   c,~
10582   caption,~
10583   cell-space-bottom-limit,~
10584   cell-space-limits,~
10585   cell-space-top-limit,~
10586   code-after,~

```

```

10587 code-for-first-col,~
10588 code-for-first-row,~
10589 code-for-last-col,~
10590 code-for-last-row,~
10591 columns-width,~
10592 corners,~
10593 custom-line,~
10594 create-extra-nodes,~
10595 create-medium-nodes,~
10596 create-large-nodes,~
10597 extra-left-margin,~
10598 extra-right-margin,~
10599 first-col,~
10600 first-row,~
10601 hlines,~
10602 hvlines,~
10603 hvlines-except-borders,~
10604 label,~
10605 last-col,~
10606 last-row,~
10607 left-margin,~
10608 light-syntax,~
10609 light-syntax-expanded,~
10610 name,~
10611 no-cell-nodes,~
10612 notes~(several~subkeys),~
10613 nullify-dots,~
10614 pgf-node-code,~
10615 renew-dots,~
10616 respect-arraystretch,~
10617 right-margin,~
10618 rounded-corners,~
10619 rules~(with~the~subkeys~'color'~and~'width'),~
10620 short-caption,~
10621 t,~
10622 tabulernote,~
10623 vlines,~
10624 xdots/color,~
10625 xdots/shorten-start,~
10626 xdots/shorten-end,~
10627 xdots/shorten~and~
10628 xdots/line-style.
10629 }

10630 \@@_msg_new:nnn { Duplicate~name }
10631 {
10632 Duplicate~name.\\
10633 The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
10634 the~same~environment~name~twice.~You~can~go~on,~but,~
10635 maybe,~you~will~have~incorrect~results~especially~
10636 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10637 message~again,~use~the~key~'allow-duplicate-names'~in~
10638 ' \token_to_str:N \NiceMatrixOptions '.\\
10639 \bool_if:NF \g_@@_messages_for_Overleaf_bool
10640 { For~a~list~of~the~names~already~used,~type~H~<return>. }
10641 }
10642 {
10643 The~names~already~defined~in~this~document~are::~
10644 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10645 }

10646 \@@_msg_new:nn { Option~auto~for~columns-width }
10647 {
10648 Erroneous~use.\\
10649 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~

```

```

10650     That~key~will~be~ignored.
10651 }
10652 \@@_msg_new:nn { NiceTabularX~without~X }
10653 {
10654     NiceTabularX~without~X.\\
10655     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10656     However,~you~can~go~on.
10657 }
10658 \@@_msg_new:nn { Preamble~forgotten }
10659 {
10660     Preamble~forgotten.\\
10661     You~have~probably~forgotten~the~preamble~of~your~
10662     \@@_full_name_env: . \\
10663     This~error~is~fatal.
10664 }
10665 \@@_msg_new:nn { Invalid~col~number }
10666 {
10667     Invalid~column~number.\\
10668     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10669     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10670 }
10671 \@@_msg_new:nn { Invalid~row~number }
10672 {
10673     Invalid~row~number.\\
10674     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10675     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10676 }
10677 \@@_define_com:NNN p ( )
10678 \@@_define_com:NNN b [ ]
10679 \@@_define_com:NNN v | |
10680 \@@_define_com:NNN V \l \l
10681 \@@_define_com:NNN B \{ \}

```

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>	36
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	193
28	Automatic arrays	194
29	The redefinition of the command <code>\dotfill</code>	195
30	The command <code>\diagbox</code>	195

31	The keyword <code>\CodeAfter</code>	197
32	The delimiters in the preamble	197
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>	213
37	The command <code>\ShowCellNames</code>	215
38	We process the options at package loading	216
39	About the package underscore	218
40	Error messages of the package	218