

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

June 25, 2026

## Abstract

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

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}
8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your~LaTeX~release~is~too~old.  \\\
11    You~need~at~least~the~version~of~2025-06-01.  \\\
12    If~you~use~Overleaf,~you~need~at~least~"TeXLive~2025".\\
13    The~package~'nicematrix'~won't~be~loaded.
14   }
15 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
16 \IfFormatAtLeastTF { 2025-06-01 }
17   { }
18 { \msg_critical:nn { nicematrix } { latex-too-old } }
```

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

```
19 \RequirePackage { amsmath }
```

---

\*This document corresponds to the version 7.10a of `nicematrix`, at the date of 2026/06/25.

```

20 \msg_new:nnn { nicematrix } { array-too-old }
21 {
22   Your~version~of~the~package~'array'~is~too~old. \\
23   You~need~at~least~the~version~of~2025-09-25. \\
24   The~package~'nicematrix'~won't~be~loaded.
25 }
26 \RequirePackage{array}
27 \IfPackageAtLeastF { array }
28 { 2025/09/25 }
29 { \msg_critical:nn { nicematrix } { array-too-old} }

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

```

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

```

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

```

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

```

44       \msg_new:nnn { nicematrix } { #1~+ } { #3 }
45     }
46   }

```

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

```

47 \cs_new_protected:Npn \@@_error_or_warning:n
48 {
49   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
50     \@@_warning:n
51     \@@_error:n
52 }

```

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”.

```

53 \bool_new:N \g_@@_messages_for_Overleaf_bool
54 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
55 {
56   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
57   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
58 }

59 \@@_msg_new:nn { mdwtab-loaded }
60 {
61   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
62   This~error~is~fatal.
63 }

```

```

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

```

We test whether the current class is revtex4-1 (deprecated) or revtex4-2 because the version 4.2f of revtex4-2 is incompatible with nicematrix.

```

66 \IfClassLoadedTF { revtex4-1 }
67 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
68 {
69   \IfClassLoadedTF { revtex4-2 }
70   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
71   {

```

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

```

72     \cs_if_exist:NT \rvtx@ifformat@geq
73     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
74     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
75   }
76 }

77 \@@_msg_new:nn { class~revtex }
78 {
79   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
80   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...\\
81   The~package~'nicematrix'~won't~be~loaded.
82 }

83 \bool_if:NT \c_@@_revtex_bool
84 { \msg_critical:nn { nicematrix } { class~revtex } }

```

## 2 Collecting options

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

*Example :*

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

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

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

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

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

```

85 \cs_new_protected:Npn \@@_collect_options:n #1
86 {
87   \peek_meaning:NTF [
88     { \@@_collect_options:nw { #1 } }
89     { #1 { } }
90   }

```

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

```

91 \NewDocumentCommand \@@_collect_options:nw { m r[] }
92 { \@@_collect_options:nn { #1 } { #2 } }
93
94 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
95 {
96   \peek_meaning:NTF [
97     { \@@_collect_options:nnw { #1 } { #2 } }
98     { #1 { #2 } }
99   }

```

```

100
101 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
102   { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

### 3 Technical definitions

Here are definitions that have been added to the LaTeX kernel in February 2006.

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

```

103 \tl_const:Nn \c_@@_c_tl { c }
104 \tl_const:Nn \c_@@_l_tl { l }
105 \tl_const:Nn \c_@@_r_tl { r }
106 \tl_const:Nn \c_@@_all_tl { all }
107 \tl_const:Nn \c_@@_dot_tl { . }
108 \str_const:Nn \c_@@_r_str { r }
109 \str_const:Nn \c_@@_c_str { c }
110 \str_const:Nn \c_@@_l_str { l }

111 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
112 \tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }

```

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

```

113 \tl_new:N \l_@@_argspec_tl

114 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
115 \cs_generate_variant:Nn \str_set:Nn { N o }
116 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
117 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
118 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
119 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
120 \cs_generate_variant:Nn \dim_min:nn { v }
121 \cs_generate_variant:Nn \dim_max:nn { v }

122 \AtBeginDocument
123   {
124     \IfPackageLoadedTF { tikz }
125     {

```

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 `\AtBeginDocument` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

126     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
127     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
128   }
129   {
130     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
131     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
132   }
133 }

```

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.

```

134 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
135 {
136   \iow_now:Nn \@mainaux
137   {
138     \ExplSyntaxOn
139     \cs_if_free:NT \pgfsyspdfmark
140     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
141     \ExplSyntaxOff
142   }
143   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
144 }

```

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.

```

145 \ProvideDocumentCommand { \iddots } { } { }
146 {
147   \mathinner
148   {
149     \mkern 1 mu
150     \box_move_up:nn { 1 pt } { \hbox { . } }
151     \mkern 2 mu
152     \box_move_up:nn { 4 pt } { \hbox { . } }
153     \mkern 2 mu
154     \box_move_up:nn { 7 pt }
155     { \vbox:n { \kern 7 pt \hbox { . } } }
156     \mkern 1 mu
157   }
158 }

```

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.

```

159 \AtBeginDocument
160 {
161   \IfPackageLoadedT { booktabs }
162   {
163     \iow_now:Nn \@mainaux
164     {
165       \ExplSyntaxOn
166       \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
167       \ExplSyntaxOff
168     }
169   }
170 }
171 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
172 {
173   \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`).

```

174   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
175   {

```

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

```

176     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
177     { \@@_old_pgful@check@rerun { ##1 } { ##2 } }
178   }
179 }

```

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

```

180 \AtBeginDocument
181 {
182   \cs_set_protected:Npe \@@_everycr:
183   {
184     \IfPackageLoadedTF { colortbl } \CT@everycr \everycr
185     { \noalign { \@@_in_everycr: } }
186   }
187   \IfPackageLoadedTF { colortbl }
188   {
189     \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
190     \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
191     \cs_new_protected:Npn \@@_revert_colortbl:
192     {
193       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
194       {
195         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
196         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
197       }
198     }
199   }

```

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).

```

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

```

204   }
205 }
206 {
207   \cs_new_protected:Npn \@@_revert_colortbl: { }
208   \cs_new_protected:Npn \@@_replace_columncolor:
209   { \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.

```

210   \def \CT@arc@ { }
211   \def \arrayrulecolor #1 # { \CT@arc { #1 } }
212   \def \CT@arc #1 #2
213   {
214     \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
215     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
216   }

```

Idem for `\CT@drs@`.

```

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

```

```

223     \def \hline
224     {
225         \noalign { \ifnum 0 = ` } \fi
226         \cs_set_eq:NN \hskip \vskip
227         \cs_set_eq:NN \vrule \hrule
228         \cs_set_eq:NN \@width \@height
229         { \CT@arc@ \vline }
230         \futurelet \reserved@a
231         \@xhline
232     }
233 }
234 }

```

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.

```

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

```

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

```

245     \skip_horizontal:N \c_zero_dim
246 }

```

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`.

```

247     \everycr { }
248     \cr
249     \noalign { \skip_vertical:n { - \arrayrulewidth } }
250 }

```

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.

```

251 \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`.

```

252 { \@@_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*.

```

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

```

---

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

```

263 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
264 {

```

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).

```

265 \int_compare:nNt { #1 } < { #2 }
266 { \multispan { \int_eval:n { #2 - #1 } } & }
267 \multispan { \int_eval:n { #3 - #2 + 1 } }
268 {
269 \CT@arc@
270 \leaders \hrule \@height \arrayrulewidth \hfill
271 \skip_horizontal:N \c_zero_dim
272 }

```

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

```

273 \peek_meaning_remove_ignore_spaces:NTF \cline
274 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
275 { \everycr { } \cr }
276 }

```

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

```

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

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

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

```

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 \tl_new:N \l_@@_tmpc_tl
317 \tl_new:N \l_@@_tmpd_tl
318 \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.

```

319 \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).

```

320 \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.

```

321 \NewExpandableDocumentCommand \NiceMatrixLastEnv { } { \int_use:N \g_@@_env_int }

```

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

```

322 \box_new:N \l_@@_the_array_box

```

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

```

323 \cs_new_protected:Npn \@@_qpoint:n #1
324 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

325 \bool_new:N \l_@@_tabular_bool

```

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

```

326 \bool_new:N \g_@@_delims_bool
327 \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.

```
328 \bool_new:N \l_@@_preamble_bool
329 \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.

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

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

```
331 \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.

```
332 \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).

```
333 \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.).

```
334 \dim_new:N \l_@@_col_width_dim
335 \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.

```
336 \int_new:N \g_@@_row_total_int
337 \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).

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

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

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

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

```
340 \tl_new:N \l_@@_hpos_cell_tl
341 \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`.

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

Idem for the mono-row blocks.

```
343 \dim_new:N \g_@@_blocks_ht_dim
344 \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}`).

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

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

```
346 \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.

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

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

```
348 \bool_new:N \l_@@_notes_detect_duplicates_bool
349 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
350 \bool_new:N \l_@@_initial_open_bool
351 \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.

```
352 \dim_new:N \l_@@_tabular_width_dim
```

`\l_@@_rule_width_before_dim` will be used before the construction of the array (that is to say during the definition of new types of rules and during the instructions used by the final user in order to require rules of several types).

```
353 \dim_new:N \l_@@_rule_width_before_dim
```

`\l_@@_rule_width_after_dim` will be used *after* the construction of the array (that is to say when we are actually drawing the rules).

```
354 \dim_new:N \l_@@_rule_width_after_dim
```

We use two variables only for legibility. We could use the same since we are not at all at the same moment in the compilation.

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

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

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

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

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

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

The following boolean will be raised when the command `\rotate` is used with the key `-90`.

```
358 \bool_new:N \g_@@_rotate_minus_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
```

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

```
360 \bool_new:N \l_@@_V_of_X_bool
```

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

```
361 \bool_new:N \g_@@_V_of_X_bool
```

```
362 \bool_new:N \g_@@_caption_finished_bool
```

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

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

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

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

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

```
365 \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 following sequence that will contain information about the size of the array.

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

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

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

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

```
369 \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).

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

For `\multicolumn`.

```
371 \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`.

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

```
373 \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 `..`.

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

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

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

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

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

```

378 \cs_new_protected:Npn \@@_test_if_math_mode:
379 {
380   \if_mode_math: \else:
381     \@@_fatal:n { Outside~math~mode }
382   \fi:
383 }

```

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

```

384 \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”.

```

385 \colorlet { nicematrix-last-col } { . }
386 \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*).

```

387 \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*.

```

388 \str_new:N \g_@@_com_or_env_str
389 \str_gset:Nn \g_@@_com_or_env_str { environment }

```

```

390 \bool_new:N \l_@@_bold_row_style_bool

```

`\g_@@_cbic_clist` is for create-blocks-in-col

```

391 \clist_new:N \g_@@_cbic_clist

```

`\g_@@_col_with_trees_clist` is for draw-trees-in-col

```

392 \clist_new:N \g_@@_col_with_trees_clist

```

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

```

393 \cs_new:Npn \@@_full_name_env:
394 {
395   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
396     { command \space \c_backslash_str \g_@@_name_env_str }
397     { environment \space \{ \g_@@_name_env_str \} }
398 }

```

```

399 \tl_new:N \g_@@_cell_after_hook_tl

```

The argument is given by curryfication.

```

400 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
401 { \tl_gput_right:Nn \g_@@_cell_after_hook_tl }

```

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

```

402 \tl_new:N \g_@@_pre_code_before_tl
403 \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`.

`\g_@@_rules_tl` will contain instructions to draw rules specified by:

- the keys `hlines` and `vlines` (and `hvlines`, `hvlines-except-borders`);
- the specifier `|` in the preamble of the argument;
- the commands `\Hline`;
- the key `hlines`, `vlines` and `hvlines` of a block;
- the key `draw` of a block;
- the command `\diagbox`.

```
404 \tl_new:N \g_@@_rules_tl
```

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

```
405 \tl_new:N \g_@@_pre_code_after_tl
406 \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.

```
407 \bool_new:N \l_@@_in_code_after_bool
```

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

```
408 \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.

```
409 \int_new:N \l_@@_old_iRow_int
410 \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).

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

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

```
412 \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.

```
413 \bool_new:N \l_@@_X_columns_aux_bool
414 \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.

```
415 \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).

```
416 \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).

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

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

```
419 \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).

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

The following dimensions will be used when drawing the dotted lines but also for the rules.

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

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

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

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

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

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

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

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

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

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

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

```
431 \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”.

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

```
433 \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.

```
434 \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!).

```
435 \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.

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

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

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

```
437 \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.

```
438 \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`).

```
439 \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`).

```
440 \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.

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

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

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

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

The following counter will be used for structures such as `\Hline\Hline`.

```
444 \int_new:N \l_@@_multiplicity_int
```

```
445 \int_set:Nn \l_@@_multiplicity_int { 1 }
```

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

```
446 \int_new:N \g_@@_ddots_int
```

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

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

```
448 \dim_new:N \g_@@_delta_x_one_dim
```

```
449 \dim_new:N \g_@@_delta_y_one_dim
```

```
450 \dim_new:N \g_@@_delta_x_two_dim
```

```
451 \dim_new:N \g_@@_delta_y_two_dim
```

---

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



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).

```

452 \int_new:N \l_@@_row_min_int
453 \int_new:N \l_@@_row_max_int
454 \int_new:N \l_@@_col_min_int
455 \int_new:N \l_@@_col_max_int

456 \int_new:N \l_@@_initial_i_int
457 \int_new:N \l_@@_initial_j_int
458 \int_new:N \l_@@_final_i_int
459 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

460 \int_new:N \l_@@_segment_start_int
461 \int_new:N \l_@@_segment_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.

```

462 \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).

```

463 \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`.

```

464 \tl_new:N \l_@@_draw_tl
465 \seq_new:N \l_@@_tikz_seq
466 \tl_new:N \l_@@_tikz_rule_tl

```

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 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`).

```

467 \str_new:N \l_@@_hpos_block_str
468 \str_set:Nn \l_@@_hpos_block_str { c }
469 \bool_new:N \l_@@_hpos_of_block_cap_bool
470 \bool_new:N \l_@@_p_block_bool

471 \bool_new:N \l_@@_fix_vertex_bool

```

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

```

472 \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).

```

473 \str_new:N \l_@@_vpos_block_str

```

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

```

474 \int_new:N \g_@@_block_box_int

```

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

```

475 \bool_new:N \l_@@_hvlines_bool

```

When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

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

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

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

## Variables for the exterior rows and columns

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

### • First row

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

```
478 \int_new:N \l_@@_first_row_int
479 \int_set:Nn \l_@@_first_row_int 1
```

### • First column

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

```
480 \int_new:N \l_@@_first_col_int
481 \int_set:Nn \l_@@_first_col_int 1
```

### • Last row

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

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

### • Last column

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

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

---

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

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

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

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

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

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

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

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

## Some utilities

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

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

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

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

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

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

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

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

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

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

```
514 \def \l_tmpa_tl { ##1 }
515 \def \l_tmpb_tl { ##1 }
516 }
517 \step_inline:nnn \l_tmpa_tl \l_tmpb_tl
518 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
```

```

519         }
520     }
521     \tl_set_eq:NN #1 \l_tmpa_clist
522 }
523 }

```

The following internal parameters are for:

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

```

524 \AtBeginDocument
525 {
526     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
527     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
528 }

```

## 5 The command `\tabularnote`

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

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

---

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

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

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

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

```
530 \int_new:N \g_@@_tabularnote_int
531 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

532 \seq_new:N \g_@@_notes_seq
533 \seq_new:N \g_@@_notes_in_caption_seq
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
546 \AtBeginDocument
547 {
548   \IfPackageLoadedTF { enumitem }
549   {
```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

```

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

```

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

```

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

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

```

## 6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

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

```

## 7 The options

The following parameter corresponds to the 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).

```

749 \bool_new:N \l_@@_caption_above_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.

```

750 \dim_new:N \l_@@_xdots_inter_dim
751 \AtBeginDocument
752 { \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).

```

753 \dim_new:N \l_@@_xdots_shorten_start_dim
754 \dim_new:N \l_@@_xdots_shorten_end_dim
755 \AtBeginDocument
756 {
757   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
758   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
759 }

```

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.

```

760 \dim_new:N \l_@@_xdots_radius_dim
761 \AtBeginDocument
762 { \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.

```

763 \tl_new:N \l_@@_xdots_line_style_tl
764 \tl_const:Nn \c_@@_standard_tl { standard }
765 \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 option `light-syntax-expanded`.

```

766 \bool_new:N \l_@@_light_syntax_bool
767 \bool_new:N \l_@@_light_syntax_expanded_bool

```

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

```

768 \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).

```

769 \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!

```

770 \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.

```

771 \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”.

```

772 \bool_new:N \l_@@_medium_nodes_bool
773 \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).

```

774 \bool_new:N \l_@@_except_borders_bool

```

```

775 \keys_define:nn { nicematrix / xdots }
776 {

```

When the key `xdots/nullify` is used, 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.).

```

777     nullify .bool_set:N = \l_@@_nullify_dots_bool ,
778     brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
779     brace-shift+ .code:n =
780       \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
781     brace-shift+ .value_required:n = true ,
782     brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
783     Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
784     shorten-start .code:n =

```

```

785 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
786 shorten-start .value_required:n = true ,
787 shorten-start+ .code:n =
788 \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
789 shorten-start~+ .meta:n = { shorten-start += #1 } ,
790 shorten-start+ .value_required:n = true ,
791 shorten-end .code:n =
792 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
793 shorten-end .value_required:n = true ,
794 shorten-end+ .code:n =
795 \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
796 shorten-end+ .value_required:n = true ,
797 shorten-end~+ .meta:n = { shorten-end += #1 } ,
798 shorten .code:n =
799 \AtBeginDocument
800 {
801 \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
802 \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
803 } ,
804 shorten .value_required:n = true ,
805 shorten+ .code:n =
806 \AtBeginDocument
807 {
808 \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
809 \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
810 } ,
811 shorten~+ .meta:n = { shorten+ = #1 } ,
812 horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
813 horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
814 line-style .code:n =
815 {
816 \bool_lazy_or:nnTF
817 { \cs_if_exist_p:N \tikzpicture }
818 { \str_if_eq_p:nn { #1 } { standard } }
819 { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
820 { \@@_error:n { bad-option-for~line-style } }
821 } ,
822 line-style .value_required:n = true ,
823 color .tl_set:N = \l_@@_xdots_color_tl ,
824 color .value_required:n = true ,
825 radius .code:n =
826 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
827 radius .value_required:n = true ,
828 inter .code:n =
829 \AtBeginDocument { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
830 radius .value_required:n = true ,

```

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

```

831 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
832 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
833 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).

```

834 draw-first .code:n = \prg_do_nothing: ,
835 unknown .code:n =
836 \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
837 }

```

```

838 \keys_define:nn { nicematrix / trees }

```

```

839 {
840   color.tl_set:N = \l_@@_trees_color_tl ,
841   color .value_required:n = true ,
842   line-width .dim_set:N = \l_@@_trees_line_width_dim ,
843   rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
844   rounded-corners .default:n = 2 pt ,
845   unknown .code:n =
846     \@@_unknown_key:nn { nicematrix / trees } { Unknown-key-for-trees }
847 }

848 \keys_define:nn { nicematrix / rules }
849 {
850   fix-vertex .code:n =
851     \bool_set_true:N \l_@@_fix_vertex_bool
852     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
853     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
854   color .tl_set:N = \l_@@_rules_color_tl ,
855   color .value_required:n = true ,
856   width .dim_set:N = \arrayrulewidth ,
857   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
858 }

```

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

```

859 \keys_define:nn { nicematrix / Global }
860 {
861   fix-vertex .value_forbidden:n = true ,
862   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
863   brace-shift+ .code:n =
864     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
865   brace-shift+ .value_required:n = true ,
866   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
867   caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
868   show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
869   color-inside .code:n = \@@_fatal:n { key-color-inside } ,
870   colortbl-like .code:n = \@@_fatal:n { key-color-inside } ,
871   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
872   &-in-blocks .meta:n = ampersand-in-blocks ,
873   no-cell-nodes .choices:nn = { true , false }
874   {
875     \tl_if_eq:NnTF \l_keys_choice_tl { true }
876     {
877       \bool_set_true:N \l_@@_no_cell_nodes_bool
878       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_inactive:
879     }
880     {
881       \bool_set_false:N \l_@@_no_cell_nodes_bool
882       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
883     }
884   } ,
885   no-cell-nodes .default:n = true ,

```

When the key `rounded-corners` is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```

886   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
887   rounded-corners .default:n = 4 pt ,
888   custom-line .code:n = \@@_custom_line:n { #1 } ,
889   default-line .code:n = \@@_default_line:n { #1 } ,
890   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
891   rules .value_required:n = true ,
892   trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
893   trees .value_required:n = true ,

```

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 `standard-cline`.

```

894   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
895   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
896   cell-space-top-limit+ .code:n =
897     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
898   cell-space-top-limit+ .value_required:n = true ,
899   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
900   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
901   cell-space-bottom-limit+ .code:n =
902     \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
903   cell-space-bottom-limit+ .value_required:n = true ,
904   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
905   cell-space-limits .meta:n =
906     {
907       cell-space-top-limit = #1 ,
908       cell-space-bottom-limit = #1 ,
909     } ,
910   cell-space-limits .value_required:n = true ,
911   cell-space-limits+ .meta:n =
912     {
913       cell-space-top-limit += #1 ,
914       cell-space-bottom-limit += #1 ,
915     } ,
916   cell-space-limits+ .value_required:n = true ,
917   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
918   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
919   light-syntax .choices:nn = { true , false }
920   {
921     \tl_if_eq:VnTF \l_keys_value_tl { true }
922     { \bool_set_true:N \l_@@_light_syntax_bool }
923     { \bool_set_false:N \l_@@_light_syntax_bool }
924     \bool_set_false:N \l_@@_light_syntax_expanded_bool
925   } ,
926   light-syntax .default:n = true ,
927   light-syntax-expanded .choices:nn = { true , false }
928   {
929     \tl_if_eq:VnTF \l_keys_value_tl { true }
930     {
931       \bool_set_true:N \l_@@_light_syntax_bool
932       \bool_set_true:N \l_@@_light_syntax_expanded_bool
933     }
934     {
935       \bool_set_false:N \l_@@_light_syntax_bool
936       \bool_set_false:N \l_@@_light_syntax_expanded_bool
937     }
938   } ,
939   light-syntax-expanded .default:n = true ,

```

The key `end-of-row` specifies the symbol used to mark the ends of rows when the light syntax is used.

```

940   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
941   end-of-row .value_required:n = true ,
942   end-of-row .initial:n = ; ,
943
944   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
945   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
946   last-row .int_set:N = \l_@@_last_row_int ,
947   last-row .default:n = -1 ,
948
949   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
950   code-for-first-col .value_required:n = true ,

```

```

951 code-for-first-col+ .code:n =
952   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
953 code-for-first-col+ .value_required:n = true ,
954 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
955
956 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
957 code-for-last-col .value_required:n = true ,
958 code-for-last-col+ .code:n =
959   { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
960 code-for-last-col+ .value_required:n = true ,
961 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
962
963 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
964 code-for-first-row .value_required:n = true ,
965 code-for-first-row+ .code:n =
966   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
967 code-for-first-row+ .value_required:n = true ,
968 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
969
970 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
971 code-for-last-row .value_required:n = true ,
972 code-for-last-row+ .code:n =
973   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
974 code-for-last-row+ .value_required:n = true ,
975 code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
976
977 hlines .clist_set:N = \l_@@_hlines_clist ,
978 hlines .default:n = all ,
979 vlines .clist_set:N = \l_@@_vlines_clist ,
980 vlines .default:n = all ,
981
982 vlines-in-sub-matrix .code:n =
983   {
984     \tl_if_single_token:nTF { #1 }
985     {
986       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
987       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

988     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
989   }
990   { \@@_error:n { One~letter~allowed } }
991 } ,
992 vlines-in-sub-matrix .value_required:n = true ,
993 hvlines .code:n =
994   {
995     \bool_set_true:N \l_@@_hvlines_bool
996     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
997     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
998   } ,
999 hvlines .value_forbidden:n = true ,
1000 hvlines-except-borders .code:n =
1001   {
1002     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1003     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1004     \bool_set_true:N \l_@@_hvlines_bool
1005     \bool_set_true:N \l_@@_except_borders_bool
1006   } ,
1007 hlines-except-borders .value_forbidden:n = true ,
1008 hlines-except-borders .code:n =
1009   {
1010     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1011     \bool_set_true:N \l_@@_hlines_bool
1012     \bool_set_true:N \l_@@_except_borders_bool

```

```

1013     } ,
1014     hlines-except-borders .value_forbidden:n = true ,
1015     parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1016     parallelize-diags .initial:n = true ,

```

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

```

1017     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1018     renew-dots .value_forbidden:n = true ,
1019     nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1020     create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1021     create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1022     create-extra-nodes .meta:n =
1023     { create-medium-nodes , create-large-nodes } ,
1024     left-margin .dim_set:N = \l_@@_left_margin_dim ,
1025     left-margin .default:n = \arraycolsep ,
1026     right-margin .dim_set:N = \l_@@_right_margin_dim ,
1027     right-margin .default:n = \arraycolsep ,
1028     margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1029     margin .default:n = \arraycolsep ,
1030     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1031     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1032     extra-margin .meta:n =
1033     { extra-left-margin = #1 , extra-right-margin = #1 } ,
1034     extra-margin .value_required:n = true ,
1035     respect-arraystretch .code:n =
1036     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1037     respect-arraystretch .value_forbidden:n = true ,

```

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

```

1038     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1039     pgf-node-code .value_required:n = true ,
1040 }

```

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

```

1041 \keys_define:nn { nicematrix / environments }
1042 {
1043     draw-trees-in-col .code:n =
1044     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1045     draw-trees-in-col .value_required:n = true ,
1046     create-blocks-in-col .code:n =
1047     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1048     create-blocks-in-col .value_required:n = true ,
1049     corners .clist_set:N = \l_@@_corners_clist ,
1050     corners .default:n = { NW , SW , NE , SE } ,
1051     code-before .code:n =
1052     {
1053         \tl_if_empty:nF { #1 }
1054         {
1055             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1056             \bool_set_true:N \l_@@_code_before_bool
1057         }
1058     } ,
1059     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}`.

```

1060     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1061     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1062     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,

```



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).

```

1063     baseline .tl_set:N = \l_@@_baseline_tl ,
1064     baseline .value_required:n = true ,
1065     baseline .initial:n = c ,
1066     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)`.

```

1067     \str_if_eq:eeTF { #1 } { auto }
1068     { \bool_set_true:N \l_@@_auto_columns_width_bool }
1069     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1070     columns-width .value_required:n = true ,
1071     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.

```

1072     \legacy_if:nF { measuring@ }
1073     {
1074         \str_set:Nc \l_@@_name_str { #1 }
1075         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1076         { \@@_err_duplicate_names:n { #1 } }
1077         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1078     } ,
1079     name .value_required:n = true ,
1080     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1081     code-after .value_required:n = true ,
1082 }

1083 \cs_set:Npn \@@_err_duplicate_names:n #1
1084 { \@@_error:nn { Duplicate-name } { #1 } }

1085 \keys_define:nn { nicematrix / notes }
1086 {
1087     no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1088     para .bool_set:N = \l_@@_notes_para_bool ,
1089     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1090     code-before .value_required:n = true ,
1091     code-before+ .code:n =
1092         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1093     code-before+ .value_required:n = true ,
1094     code-before~+ .code:n =
1095         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1096     code-before~+ .value_required:n = true ,
1097     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1098     code-after .value_required:n = true ,
1099     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1100     style .cs_set:Np = \@@_notes_style:n #1 ,
1101     style .value_required:n = true ,
1102     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1103     label-in-tabular .value_required:n = true ,
1104     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1105     label-in-list .value_required:n = true ,
1106     enumitem-keys .code:n =
1107     {
1108         \AtBeginDocument
1109         {
1110             \IfPackageLoadedT { enumitem }
1111             { \setlist* [ tabularnotes ] { #1 } }
1112         }
1113     } ,
1114     enumitem-keys .value_required:n = true ,
1115     enumitem-keys-para .code:n =
1116     {

```

```

1117 \AtBeginDocument
1118 {
1119     \IfPackageLoadedT { enumitem }
1120     { \setlist* [ tabularnotes* ] { #1 } }
1121 }
1122 } ,
1123 enumitem-keys-para .value_required:n = true ,
1124 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1125 unknown .code:n =
1126     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1127 }

```

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.

```

1128 \keys_define:nn { nicematrix / delimiters }
1129 {
1130     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1131     color .tl_set:N = \l_@@_delimiters_color_tl ,
1132     color .value_required:n = true ,
1133 }

```

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

```

1134 \keys_define:nn { nicematrix }
1135 {
1136     NiceMatrixOptions .inherit:n =
1137     { nicematrix / Global } ,
1138     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1139     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1140     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1141     NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1142     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1143     SubMatrix / rules .inherit:n = nicematrix / rules ,
1144     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1145     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1146     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1147     NiceMatrix .inherit:n =
1148     {
1149         nicematrix / Global ,
1150         nicematrix / environments ,
1151     } ,
1152     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1153     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1154     NiceMatrix / trees .inherit:n = nicematrix / trees ,
1155     NiceTabular .inherit:n =
1156     {
1157         nicematrix / Global ,
1158         nicematrix / environments
1159     } ,
1160     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1161     NiceTabular / rules .inherit:n = nicematrix / rules ,
1162     NiceTabular / notes .inherit:n = nicematrix / notes ,
1163     NiceTabular / trees .inherit:n = nicematrix / trees ,
1164     NiceArray .inherit:n =
1165     {
1166         nicematrix / Global ,
1167         nicematrix / environments ,

```

```

1168     } ,
1169     NiceArray / xdots .inherit:n = nicematrix / xdots ,
1170     NiceArray / rules .inherit:n = nicematrix / rules ,
1171     NiceArray / trees .inherit:n = nicematrix / trees ,
1172     pNiceArray .inherit:n =
1173     {
1174         nicematrix / Global ,
1175         nicematrix / environments ,
1176     } ,
1177     pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1178     pNiceArray / rules .inherit:n = nicematrix / rules ,
1179     pNiceArray / trees .inherit:n = nicematrix / trees
1180 }

```

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

```

1181 \keys_define:nn { nicematrix / NiceMatrixOptions }
1182 {
1183     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1184     delimiters / color .value_required:n = true ,
1185     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1186     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1187     delimiters .value_required:n = true ,
1188     width .dim_set:N = \l_@@_width_dim ,
1189     last-col .code:n =
1190     \tl_if_empty:nF { #1 }
1191     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1192     \int_zero:N \l_@@_last_col_int ,
1193     small .bool_set:N = \l_@@_small_bool ,
1194     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.

```

1195     renew-matrix .code:n = \@@_renew_matrix: ,
1196     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}`.

```

1197     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.

```

1198     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)`.

```

1199     \str_if_eq:eeTF { #1 } { auto }
1200     { \@@_error:n { Option~auto~for~columns-width } }
1201     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1202     allow-duplicate-names .code:n =
1203     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1204     allow-duplicate-names .value_forbidden:n = true ,
1205     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1206     notes .value_required:n = true ,
1207     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1208     sub-matrix .value_required:n = true ,
1209     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,

```

```

1210 matrix / columns-type .value_required:n = true ,
1211 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1212 unknown .code:n =
1213   \@@_unknown_key:nn
1214   { nicematrix / Global , nicematrix / NiceMatrixOptions }
1215   { Unknown-key-for-NiceMatrixOptions }
1216 }

```

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

```

1217 \NewDocumentCommand \NiceMatrixOptions { m }
1218 { \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.

```

1219 \keys_define:nn { nicematrix / NiceMatrix }
1220 {
1221   last-col .code:n = \tl_if_empty:nTF { #1 }
1222   {
1223     \bool_set_true:N \l_@@_last_col_without_value_bool
1224     \int_set:Nn \l_@@_last_col_int { -1 }
1225   }
1226   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1227   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1228   columns-type .value_required:n = true ,
1229   l .meta:n = { columns-type = l } ,
1230   r .meta:n = { columns-type = r } ,
1231   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1232   delimiters / color .value_required:n = true ,
1233   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1234   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1235   delimiters .value_required:n = true ,
1236   small .bool_set:N = \l_@@_small_bool ,
1237   small .value_forbidden:n = true ,
1238   unknown .code:n =
1239     \@@_unknown_key:nn
1240     { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1241     { Unknown-key-for-NiceMatrix }
1242 }

```

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

```

1243 \keys_define:nn { nicematrix / NiceArray }
1244 {

```

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.

```

1245   small .bool_set:N = \l_@@_small_bool ,
1246   small .value_forbidden:n = true ,
1247   last-col .code:n = \tl_if_empty:nF { #1 }
1248     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1249     \int_zero:N \l_@@_last_col_int ,
1250   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1251   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1252   unknown .code:n =
1253     \@@_unknown_key:nn
1254     { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1255     { Unknown-key-for-NiceArray }
1256 }

```

```

1257 \keys_define:nn { nicematrix / pNiceArray }
1258 {
1259   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1260   last-col .code:n = \tl_if_empty:nF { #1 }
1261     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1262     \int_zero:N \l_@@_last_col_int ,
1263   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1264   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1265   delimiters / color .value_required:n = true ,
1266   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1267   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1268   delimiters .value_required:n = true ,
1269   small .bool_set:N = \l_@@_small_bool ,
1270   small .value_forbidden:n = true ,
1271   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1272   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1273   unknown .code:n =
1274     \@@_unknown_key:nn
1275     { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1276     { Unknown-key-for-NiceMatrix }
1277 }

```

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

```

1278 \keys_define:nn { nicematrix / NiceTabular }
1279 {

```

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.

```

1280   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1281     \bool_set_true:N \l_@@_width_used_bool ,
1282   width .value_required:n = true ,
1283   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1284   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1285   tabularnote .value_required:n = true ,
1286   caption .tl_set:N = \l_@@_caption_tl ,
1287   caption .value_required:n = true ,
1288   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1289   short-caption .value_required:n = true ,
1290   label .tl_set:N = \l_@@_label_tl ,
1291   label .value_required:n = true ,
1292   last-col .code:n = \tl_if_empty:nF { #1 }
1293     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1294     \int_zero:N \l_@@_last_col_int ,
1295   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1296   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1297   unknown .code:n =
1298     \@@_unknown_key:nn
1299     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1300     { Unknown-key-for-NiceTabular }
1301 }

```

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
```

```

1302 \keys_define:nn { nicematrix / CodeAfter }
1303 {
1304   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1305   delimiters / color .value_required:n = true ,

```

```

1306     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1307     rules .value_required:n = true ,
1308     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1309     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1310     sub-matrix .value_required:n = true ,
1311     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1312 }

```

## 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}`).

```

1313 \cs_new_protected:Npn \@@_cell_begin:
1314 {

```

`\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.

```

1315     \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).

```

1316     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1317     \cs_set_eq:NN \Hline \@@_Hline_in_cell:

```

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

```

1318     \int_gincr:N \c@jCol

```

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

Here is a version with the standard syntax of L3.

```

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

```

We will use a version a little more efficient.

```

1319     \if_int_compare:w \c@jCol = \c_one_int
1320     \if_int_compare:w \l_@@_first_col_int = \c_one_int
1321     \@@_begin_of_row:
1322     \fi:
1323     \fi:

```

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

```

1324     \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1325     \@@_tuning_not_tabular_begin:
1326     \@@_tuning_exterior_rows:
1327     \g_@@_row_style_tl
1328 }
1329 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }

```

Here is a version with the standard syntax of the L3 programming layer.

```

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

```

We will use a version a little more efficient.

```

1330 \cs_new_protected:Npn \@@_tuning_first_last_row:
1331 {
1332   \if_int_compare:w \c@iRow = \c_zero_int
1333   \if_int_compare:w \c@jCol > \c_zero_int
1334   \l_@@_code_for_first_row_tl
1335   \@@_tuning_key_small: % 2026/04/06
1336   \xglobal \colorlet { nicematrix-first-row } { . }
1337   \fi:
1338   \else:
1339   \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1340   \fi:
1341 }

```

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.

```

1342 \cs_new_protected:Npn \@@_tuning_last_row:
1343 {
1344   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1345   \l_@@_code_for_last_row_tl
1346   \@@_tuning_key_small: % 2026/04/06
1347   \xglobal \colorlet { nicematrix-last-row } { . }
1348   \fi:
1349 }

```

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

```

1350 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1351 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1352 {
1353   \m@th
1354   $ % $

```

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

```

1355   \@@_tuning_key_small:
1356 }
1357 \cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $

```

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

```

1358 \cs_new_protected:Npn \@@_begin_of_row:
1359 {
1360   \int_gincr:N \c@iRow
1361   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1362   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1363   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1364   \pgfpicture
1365   \pgfrememberpicturepositiononpagetrue
1366   \pgfcoordinate
1367   { \@@_env: - row - \int_use:N \c@iRow - base }
1368   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1369   \str_if_empty:NF \l_@@_name_str
1370   {
1371     \pgfnodealias
1372     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1373     { \@@_env: - row - \int_use:N \c@iRow - base }
1374   }
1375   \endpgfpicture
1376 }

```

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

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

```

\cs_new_protected:Npn \@@_update_for_first_and_last_row:
{
  \int_if_zero:nTF { \c@iRow }
  {
    \dim_compare:nNnT
    { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
    { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
    \dim_compare:nNnT
    { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
    { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
  }
  {
    \int_compare:nNnT { \c@iRow } = { 1 }
    {
      \dim_compare:nNnT
      { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
      { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
    }
  }
}

```

We will use a version a little more efficient.

```

1377 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1378 {
1379   \if_int_compare:w \c@iRow = \c_zero_int
1380     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1381       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1382     \fi:
1383     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1384       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1385     \fi:
1386   \else:
1387     \if_int_compare:w \c@iRow = \c_one_int

```



```

1388         \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1389         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1390     \fi:
1391 \fi:
1392 \fi:
1393 }

1394 \cs_new_protected:Npn \@@_rotate_cell_box:
1395 {
1396     \box_rotate:Nn \l_@@_cell_box
1397     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1398     \bool_if:NTF \g_@@_rotate_c_bool
1399     {
1400         \hbox_set:Nn \l_@@_cell_box
1401         {
1402             \IfFormatAtLeastTF { 2026-04-01 }
1403             { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1404             {
1405                 \m@th
1406                 $ % $
1407                 \vcenter { \box_use:N \l_@@_cell_box }
1408                 $ % $
1409             }
1410         }
1411     }
1412     {
1413         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1414         {
1415             \vbox_set_top:Nn \l_@@_cell_box
1416             {
1417                 \vbox_to_zero:n { }
1418                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1419                 \box_use:N \l_@@_cell_box
1420             }
1421         }
1422     }
1423     \bool_gset_false:N \g_@@_rotate_bool
1424     \bool_gset_false:N \g_@@_rotate_c_bool
1425     \bool_gset_false:N \g_@@_rotate_minus_bool
1426 }

```

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

```

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

```

Here is a version slightly more efficient.

```

1427 \cs_set_protected:Npn \@@_adjust_size_box:
1428 {
1429   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1430     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1431       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1432     \fi:
1433     \global \g_@@_blocks_wd_dim = \c_zero_dim
1434   \fi:
1435   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1436     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1437       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1438     \fi:
1439     \global \g_@@_blocks_dp_dim = \c_zero_dim
1440   \fi:
1441   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1442     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1443       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1444     \fi:
1445     \global \g_@@_blocks_ht_dim = \c_zero_dim
1446   \fi:
1447 }
1448 \cs_new_protected:Npn \@@_cell_end:
1449 {

```

The following command is nullified in the tabulars.

```

1450   \@@_tuning_not_tabular_end:
1451   \hbox_set_end:
1452   \@@_cell_end_i:
1453 }

```

```

\cs_new_protected:Npn \@@_cell_end_i:
{
  \g_@@_cell_after_hook_tl
  \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
  \@@_adjust_size_box:
  \box_set_ht:Nn \l_@@_cell_box
  { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
  \box_set_dp:Nn \l_@@_cell_box
  { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
  \@@_update_max_cell_width:
  \@@_update_for_first_and_last_row:
  \bool_if:NTF \g_@@_empty_cell_bool
  { \box_use_drop:N \l_@@_cell_box }
  {
    \bool_if:NTF \g_@@_not_empty_cell_bool
    { \@@_print_node_cell: }
    {
      \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
      { \@@_print_node_cell: }
      { \box_use_drop:N \l_@@_cell_box }
    }
  }
  \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
  { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
  \bool_gset_false:N \g_@@_empty_cell_bool
  \bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1454 \cs_new_protected:Npn \@@_cell_end_i:
1455 {

```

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.

```

1456   \g_@@_cell_after_hook_tl
1457   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1458   \@@_adjust_size_box:
1459   \box_set_ht:Nn \l_@@_cell_box
1460     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1461   \box_set_dp:Nn \l_@@_cell_box
1462     { \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”).

```

1463   \@@_update_max_cell_width:

```

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

```

1464   \@@_update_for_first_and_last_row:

```

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

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

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

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

```

1465   \bool_if:NTF \g_@@_empty_cell_bool
1466     { \box_use_drop:N \l_@@_cell_box }
1467   {
1468     \bool_if:NTF \g_@@_not_empty_cell_bool
1469       \@@_print_node_cell:
1470       {
1471         \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1472           \@@_print_node_cell:
1473         \else:
1474           \box_use_drop:N \l_@@_cell_box
1475         \fi:
1476       }
1477   }
1478   \if_int_compare:w \c@jCol > \g_@@_col_total_int
1479     \global \g_@@_col_total_int = \c@jCol
1480   \fi:
1481   \global \let \g_@@_empty_cell_bool \c_false_bool
1482   \global \let \g_@@_not_empty_cell_bool \c_false_bool
1483 }

```

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

```

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

```

We will use the following version, slightly more efficient:

```

1484 \cs_new_protected:Npn \@@_update_max_cell_width:
1485 {
1486   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1487     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1488   \fi:
1489 }

```

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`).

```

1490 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1491 {
1492   \@@_math_toggle:
1493   \hbox_set_end:
1494   \bool_if:NF \g_@@_rotate_bool
1495   {
1496     \hbox_set:Nn \l_@@_cell_box
1497     {
1498       \makebox [ \l_@@_col_width_dim ] [ s ]
1499       { \hbox_unpack_drop:N \l_@@_cell_box }
1500     }
1501   }
1502   \@@_cell_end_i:
1503 }

```

```

1504 \pgfset
1505 {
1506   nicematrix / cell-node /.style =
1507   {
1508     inner~sep = \c_zero_dim ,
1509     minimum~width = \c_zero_dim
1510   }
1511 }

```

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.

```

1512 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1513 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1514 {
1515   \use:c
1516   {
1517     __siunitx_table_align_
1518     \bool_if:NTF \l__siunitx_table_text_bool
1519       \l__siunitx_table_align_text_tl
1520       \l__siunitx_table_align_number_str
1521     :n
1522   }
1523   { #1 }
1524 }

```

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!

```

1525 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1526 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1527 {
1528   \box_move_up:nn { \box_ht:N \l_@@_cell_box }

```

```

1529 \hbox:n
1530 {
1531   \pgfsys@markposition
1532   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1533 }
1534 #1
1535 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1536 \hbox:n
1537 {
1538   \pgfsys@markposition
1539   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1540 }
1541 }

1542 \cs_new_protected:Npn \@@_print_node_cell:
1543 {
1544   \socket_use:nn { nicematrix / siunitx-wrap }
1545   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1546 }

```

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

```

1547 \cs_new_protected:Npn \@@_node_cell_active:
1548 {
1549   \pgfpicture
1550   \pgfsetbaseline \c_zero_dim
1551   \pgfrememberpicturepositiononpagetrue
1552   \pgfset { nicematrix / cell-node }
1553   \pgfnode
1554   { rectangle }
1555   { base }
1556   {

```

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).

```

1557 \sys_if_engine_xetex:T { \set@color }
1558 \box_use:N \l_@@_cell_box
1559 }
1560 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1561 { \l_@@_pgf_node_code_tl }
1562 \str_if_empty:NF \l_@@_name_str
1563 {
1564   \pgfnodealias
1565   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1566   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1567 }
1568 \endpgfpicture
1569 }
1570 \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1571 \cs_new_protected:Npn \@@_node_cell_inactive:
1572 { \set@color \box_use_drop:N \l_@@_cell_box }

```

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).

```
1573 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3  
1574 {  
1575   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce  
1576   { g_@@_ #2 _ lines _ tl }  
1577   {  
1578     \use:c { @@ _ draw _ #2 : nnn }  
1579     { \int_use:N \c@iRow }  
1580     { \int_use:N \c@jCol }  
1581     { \exp_not:n { #3 } }  
1582   }  
1583 }  
  
1584 \cs_new_protected:Npn \@@_array:n  
1585 {  
1586   \dim_set:Nn \col@sep  
1587   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }  
1588   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim  
1589   { \def \@halignto { } }  
1590   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1591   \@tabarray  
  
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose  
the \array (of array) with the option t and the right translation will be done further. Re-  
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.  
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).  
1592   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]  
1593 }  
1594 \cs_generate_variant:Nn \@@_array:n { o }
```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. 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`.

```
1595 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1596 \cs_new_protected:Npn \@@_create_row_node:  
1597 {  
1598   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int  
1599   {  
1600     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow  
1601     \@@_create_row_node_i:  
1602   }  
1603 }  
  
1604 \cs_new_protected:Npn \@@_create_row_node_i:  
1605 {
```

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

```
1606   \hbox  
1607   {  
1608     \bool_if:NT \l_@@_code_before_bool  
1609     {  
1610       \vtop
```

```

1611         {
1612             \skip_vertical:N 0.5\arrayrulewidth
1613             \pgfsys@markposition
1614             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1615             \skip_vertical:N -0.5\arrayrulewidth
1616         }
1617     }
1618     \pgfpicture
1619     \pgfrememberpicturepositiononpagetrue
1620     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1621     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1622     \str_if_empty:NF \l_@@_name_str
1623     {
1624         \pgfnodealias
1625         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1626         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1627     }
1628     \endpgfpicture
1629 }
1630 }

```

```

1631 \cs_new_protected:Npn \@@_in_everycr:
1632 {
1633     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1634     \tbl_update_cell_data_for_next_row:
1635     \int_gzero:N \c@jCol
1636     \bool_gset_false:N \g_@@_after_col_zero_bool
1637     \bool_if:NF \g_@@_row_of_col_done_bool
1638     {
1639         \@@_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).

```

1640     \clist_if_empty:NF \l_@@_hlines_clist
1641     {
1642         \str_if_eq:eeF \l_@@_hlines_clist { all }
1643         {
1644             \clist_if_in:NeT
1645             \l_@@_hlines_clist
1646             { \int_eval:n { \c@iRow + 1 } }
1647         }
1648     }

```

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.

```

1649         \int_compare:nNnT \c@iRow > { -1 }
1650         {
1651             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1652             { \hrule height \arrayrulewidth width \c_zero_dim }
1653         }
1654     }
1655 }
1656 }
1657 }

```

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

```

1658 \cs_set_protected:Npn \@@_renew_dots:
1659 {
1660     \cs_set_eq:NN \ldots \@@_Ldots:
1661     \cs_set_eq:NN \cdots \@@_Cdots:
1662     \cs_set_eq:NN \vdots \@@_Vdots:
1663     \cs_set_eq:NN \ddots \@@_Ddots:

```

```

1664 \cs_set_eq:NN \iddots \@@_Iddots:
1665 \cs_set_eq:NN \dots \@@_Ldots:
1666 \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1667 }

```

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's why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition <sup>5</sup>.

```

1668 \AtBeginDocument
1669 {
1670   \IfPackageLoadedTF { booktabs }
1671   {
1672     \cs_new_protected:Npn \@@_patch_booktabs:
1673     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1674   }
1675   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1676 }

```

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

```

1677 \cs_new_protected:Npn \@@_some_initialization:
1678 {
1679   \@@_everycr:
1680   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1681   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1682   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1683   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1684   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1685   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1686 }

```

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

```

1687 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1688 {

```

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

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

```

1689   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1690   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1691   \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

---

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

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



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.

```
1692 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1693 \fp_gzero:N \g_@@_total_X_weight_fp
1694 \bool_gset_false:N \g_@@_V_of_X_bool
1695 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1696 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1697 \@@_patch_booktabs:
1698 \box_clear_new:N \l_@@_cell_box
1699 \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}`).

```
1700 \bool_if:NT \l_@@_small_bool
1701 {
1702   \def \arraystretch { 0.47 }
1703   \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1704 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1705 }
```

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!

```
1706 \bool_if:NT \g_@@_create_cell_nodes_bool
1707 {
1708   \tbl_put_right:Nn \@@_begin_of_row:
1709   {
1710     \pgfsys@markposition
1711     { \@@_env: - row - \int_use:N \c@iRow - base }
1712   }
1713   \socket_assign_plugin:nn { nicematrix / create-cell-nodes } { active }
1714 }
```

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

```
1715 \def \ar@ialign
1716 {
1717   \tbl_init_cell_data_for_table:
1718   \@@_some_initialization:
1719   \dim_zero:N \tabskip
```

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

```
1720 \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1721 \halign
1722 }
```

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).

```

1723 \cs_set_eq:NN \@@_old_ldots: \ldots
1724 \cs_set_eq:NN \@@_old_cdots: \cdots
1725 \cs_set_eq:NN \@@_old_vdots: \vdots
1726 \cs_set_eq:NN \@@_old_ddots: \ddots
1727 \cs_set_eq:NN \@@_old_iddots: \iddots
1728 \bool_if:NTF \l_@@_standard_cline_bool
1729 { \cs_set_eq:NN \cline \@@_standard_cline: }
1730 { \cs_set_eq:NN \cline \@@_cline: }
1731 \cs_set_eq:NN \Ldots \@@_Ldots:
1732 \cs_set_eq:NN \Cdots \@@_Cdots:
1733 \cs_set_eq:NN \Vdots \@@_Vdots:
1734 \cs_set_eq:NN \Ddots \@@_Ddots:
1735 \cs_set_eq:NN \Iddots \@@_Iddots:
1736 \cs_set_eq:NN \Hline \@@_Hline:
1737 \cs_set_eq:NN \Hspace \@@_Hspace:
1738 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1739 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1740 \cs_set_eq:NN \Block \@@_Block:
1741 \cs_set_eq:NN \rotate \@@_rotate:
1742 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1743 \cs_set_eq:NN \dotfill \@@_dotfill:
1744 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1745 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1746 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1747 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1748 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1749 \cs_set_eq:NN \TopRule \@@_TopRule
1750 \cs_set_eq:NN \MidRule \@@_MidRule
1751 \cs_set_eq:NN \BottomRule \@@_BottomRule
1752 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1753 \cs_set_eq:NN \Hbrace \@@_Hbrace
1754 \cs_set_eq:NN \Vbrace \@@_Vbrace
1755 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1756 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1757 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1758 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1759 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1760 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1761 \int_if_zero:nTF \l_@@_first_row_int
1762 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1763 {
1764 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1765 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1766 }
1767 \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:`.

```

1768 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1769 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1770 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1771 \@@_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!).

```

1772 \tl_if_exist:NT \l_@@_note_in_caption_tl

```

```

1773 {
1774   \tl_if_empty:NF \l_@@_note_in_caption_tl
1775   {
1776     \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1777     \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1778   }
1779 }

```

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

```

1780 \seq_gclear:N \g_@@_multicolumn_cells_seq
1781 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

When we compose a cell which belongs to a column which contains a instruction `\columncolor` (in the preamble of the environment), we add the number of that column in the following sequence (in order to recall that we have written the following instruction of the `\g_@@_pre_code_before_tl`).

```

1782 \seq_gclear_new:N \g_@@_columncolor_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).

```

1783 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

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

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

```

1784 \int_gzero:N \g_@@_row_total_int

```

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

```

1785 \int_gzero:N \g_@@_col_total_int
1786 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1787 \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.

```

1788 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1789 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1790 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1791 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1792 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1793 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1794 \tl_gclear:N \g_nicematrix_code_before_tl
1795 \tl_gclear:N \g_@@_pre_code_before_tl

```

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

```

1796 \dim_zero_new:N \l_@@_left_delim_dim
1797 \dim_zero_new:N \l_@@_right_delim_dim
1798 \bool_if:NTF \g_@@_delims_bool
1799 {

```

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

```

1800 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1801 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1802 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1803 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1804 }
1805 {
1806   \dim_gset:Nn \l_@@_left_delim_dim

```

```

1807         { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1808         \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1809     }
1810 }

```

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

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

```

1811 \cs_new_protected:Npn \@@_pre_array:
1812 {
1813     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1814     \int_gzero_new:N \c@iRow
1815     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1816     \int_gzero_new:N \c@jCol

```

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

```

1817     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1818     { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1819     \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1820     { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1821     \bool_if:NT \g_@@_aux_found_bool
1822     {
1823         \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1824         \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1825         \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1826         \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1827     }

```

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

```

1828     \int_compare:nNnT \l_@@_last_row_int = { -1 }
1829     {
1830         \bool_set_true:N \l_@@_last_row_without_value_bool
1831         \bool_if:NT \g_@@_aux_found_bool
1832         { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1833     }
1834     \int_compare:nNnT \l_@@_last_col_int = { -1 }
1835     {
1836         \bool_if:NT \g_@@_aux_found_bool
1837         { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1838     }

```

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”.

```

1839     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1840     {
1841         \tl_put_right:Nn \@@_update_for_first_and_last_row:
1842         {
1843             \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1844             { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1845             \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1846             { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1847         }
1848     }

```

```

1849 \seq_gclear:N \g_@@_cols_vlism_seq
1850 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1851 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1852 \@@_pre_array_after_CodeBefore:

```

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

```

1853 \hbox_set:Nw \l_@@_the_array_box
1854 \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1855 \skip_horizontal:N \l_@@_extra_left_margin_dim
1856 \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).

```

1857 \m@th
1858 $ % $
1859 \bool_if:NTF \l_@@_light_syntax_bool
1860 { \use:c { @@-light-syntax } }
1861 { \use:c { @@-normal-syntax } }
1862 }

```

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

```

1863 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1864 {
1865   \tl_set:Nn \l_tmpa_tl { #1 }
1866   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1867   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1868   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1869   \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...

```

1870 \@@_pre_array:
1871 }

```

## 9 The `\CodeBefore`

```

1872 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1873 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Bad~use~of~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).

```

1874 \cs_new_protected:Npn \@@_pre_code_before:
1875 {

```

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

```

1876 \pgfsys@markposition { \@@_env: - position }
1877 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1878 \pgfpicture
1879 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1880 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1881 {
1882   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1883   \pgfcoordinate { \@@_env: - row - ##1 }
1884   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1885 }

```

Now, the recreation of the col nodes.

```

1886 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1887 {
1888   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1889   \pgfcoordinate { \@@_env: - col - ##1 }
1890   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1891 }

```

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

```

1892 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1893 \endpgfpicture

```

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

```

1894 \@@_create_diag_nodes:

```

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

```

1895 \@@_create_blocks_nodes:
1896 \IfPackageLoadedT { tikz }
1897 {
1898   \tikzset
1899   {
1900     every-picture / .style =
1901     { overlay , name-prefix = \@@_env: - }
1902   }
1903 }
1904 \cs_set_eq:NN \cellcolor \@@_cellcolor
1905 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1906 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1907 \cs_set_eq:NN \rowcolor \@@_rowcolor
1908 \cs_set_eq:NN \rowcolors \@@_rowcolors
1909 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1910 \cs_set_eq:NN \arraycolor \@@_arraycolor
1911 \cs_set_eq:NN \columncolor \@@_columncolor
1912 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1913 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1914 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1915 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1916 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1917 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1918 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1919 }

1920 \cs_new_protected:Npn \@@_exec_code_before:
1921 {

```

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 detect whether we actually have coloring instructions to execute...

```

1922 \clist_map_inline:Nn \l_@@_corners_cells_clist
1923 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1924 \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`.

```

1925 \@@_add_to_colors_seq:nn { { nocolor } } { }
1926 \bool_gset_false:N \g_@@_create_cell_nodes_bool

```

```
1927 \group_begin:
```

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

```
1928 \if_mode_math:
1929   \@@_exec_code_before_i:
1930 \else:
1931   $ % $
1932   \@@_exec_code_before_i:
1933   $ % $
1934 \fi:
1935 \group_end:
1936 }
```

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`).

```
1937 \cs_new_protected:Npn \@@_exec_code_before_i:
1938 {
1939   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1940   { \@@_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`.

```
1941 \exp_last_unbraced:No \@@_CodeBefore_keys:
1942 \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.

```
1943 \@@_actually_color:
1944 \l_@@_code_before_tl
1945 \q_stop
1946 }
```

```
1947 \keys_define:nn { nicematrix / CodeBefore }
1948 {
1949   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1950   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1951   sub-matrix .value_required:n = true ,
1952   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1953   delimiters / color .value_required:n = true ,
1954   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1955 }
1956 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1957 {
1958   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1959   \@@_CodeBefore:w
1960 }
```

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.

```
1961 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1962 {
1963   \bool_if:NTF \g_@@_aux_found_bool
1964   {
1965     \@@_pre_code_before:
1966     \legacy_if:nF { measuring@ } { #1 }
1967   }
```

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

```

1968     {
1969         \pgfsys@markposition { \@@_env: - position }
1970         \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1971         \pgfpicture
1972         \pgf@relevantforpicturesizefalse
1973         \endpgfpicture

```

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

```

1974         \pgfpicture
1975         \pgfrememberpicturepositiononpagetrue
1976         \endpgfpicture

```

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

```

1977         \pgfpicture
1978         \pgf@relevantforpicturesizefalse
1979         \pgfrememberpicturepositiononpagetrue
1980         \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1981         \pgfpicture
1982         \pgf@relevantforpicturesizefalse
1983         \endpgfpicture
1984     }
1985 }

```

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.

```

1986 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1987 {
1988     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1989     {
1990         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1991         \pgfcoordinate { \@@_env: - row - ##1 - base }
1992         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1993         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1994         {
1995             \cs_if_exist:cT
1996             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1997             {
1998                 \pgfsys@getposition
1999                 { \@@_env: - ##1 - #####1 - NW }
2000                 \@@_node_position:
2001                 \pgfsys@getposition
2002                 { \@@_env: - ##1 - #####1 - SE }
2003                 \@@_node_position_i:
2004                 \@@_pgf_rect_node:nnn
2005                 { \@@_env: - ##1 - #####1 }
2006                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2007                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2008             }
2009         }
2010     }
2011     \@@_create_extra_nodes:
2012     \@@_create_aliases_last:
2013 }

2014 \cs_new_protected:Npn \@@_create_aliases_last:
2015 {
2016     \int_step_inline:nn \c@iRow

```



```

2017 {
2018   \pgfnodealias
2019   { \@@_env: - ##1 - last }
2020   { \@@_env: - ##1 - \int_use:N \c@jCol }
2021 }
2022 \int_step_inline:nn \c@jCol
2023 {
2024   \pgfnodealias
2025   { \@@_env: - last - ##1 }
2026   { \@@_env: - \int_use:N \c@iRow - ##1 }
2027 }
2028 \pgfnodealias
2029 { \@@_env: - last - last }
2030 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2031 }

```

```

2032 \cs_new_protected:Npn \@@_create_blocks_nodes:
2033 {
2034   \pgfpicture
2035   \pgf@relevantforpicturesizefalse
2036   \pgfrememberpicturepositiononpagetrue
2037   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2038   { \@@_create_one_block_node:nnnnn ##1 }
2039   \endpgfpicture
2040 }

```

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

```

2041 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2042 {
2043   \tl_if_empty:nF { #5 }
2044   {
2045     \@@_qpoint:n { col - #2 }
2046     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2047     \@@_qpoint:n { #1 }
2048     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2049     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2050     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2051     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2052     \@@_pgf_rect_node:nnnnn
2053     { \@@_env: - #5 }
2054     { \dim_use:N \l_tmpa_dim }
2055     { \dim_use:N \l_tmpb_dim }
2056     { \dim_use:N \l_@@_tmpc_dim }
2057     { \dim_use:N \pgf@y }
2058   }
2059 }

```

## 10 The environment `{NiceArrayWithDelims}`

```

2060 \NewDocumentEnvironment { NiceArrayWithDelims }
2061 { m m O { } m ! O { } t \CodeBefore }
2062 {
2063   \@@_provide_pgfsyspdfmark:
2064   \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 exponent to a matrix in a mathematical formula.

---

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

```

2065 \bgroup

2066 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2067 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2068 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2069 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2070 \int_gzero:N \g_@@_block_box_int
2071 \dim_gzero:N \g_@@_width_last_col_dim
2072 \dim_gzero:N \g_@@_width_first_col_dim
2073 \bool_gset_false:N \g_@@_row_of_col_done_bool
2074 \str_if_empty:NT \g_@@_name_env_str
2075 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2076 \bool_if:NTF \l_@@_tabular_bool
2077 \mode_leave_vertical:
2078 \@@_test_if_math_mode:
2079 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2080 \bool_set_true:N \l_@@_in_env_bool

```

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

```

2081 \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.

```

2082 \cs_if_exist:NT \tikz@library@external@loaded
2083 {
2084   \tikzexternaldisable
2085   \cs_if_exist:NT \ifstandalone
2086   { \tikzset { external / optimize = false } }
2087 }

```

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

```

2088 \int_gincr:N \g_@@_env_int
2089 \bool_if:NF \l_@@_block_auto_columns_width_bool
2090 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

2091 \seq_gclear:N \g_@@_blocks_seq
2092 \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`.

```

2093 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2094 \seq_gclear:N \g_@@_pos_of_xdots_seq
2095 \tl_gclear_new:N \g_@@_code_before_tl
2096 \tl_gclear:N \g_@@_row_style_tl

```

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

```

2097 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2098 {
2099   \bool_gset_true:N \g_@@_aux_found_bool
2100   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2101 }
2102 { \bool_gset_false:N \g_@@_aux_found_bool }

```

---

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

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.

```

2103 \tl_gclear:N \g_@@_aux_tl
2104 \tl_if_empty:NF \g_@@_code_before_tl
2105 {
2106   \bool_set_true:N \l_@@_code_before_bool
2107   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2108 }
2109 \tl_if_empty:NF \g_@@_pre_code_before_tl
2110 { \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.

```

2111 \bool_if:NTF \g_@@_delims_bool
2112 { \keys_set:nn { nicematrix / pNiceArray } }
2113 { \keys_set:nn { nicematrix / NiceArray } }
2114 { #3 , #5 }

2115 \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302

```

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

```

2116 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2117 }

```

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

```

2118 {
2119   \bool_if:NTF \l_@@_light_syntax_bool
2120   { \use:c { end @@-light-syntax } }
2121   { \use:c { end @@-normal-syntax } }
2122   $ % $
2123   \skip_horizontal:N \l_@@_right_margin_dim
2124   \skip_horizontal:N \l_@@_extra_right_margin_dim
2125   \hbox_set_end:
2126   \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.

```

2127 \bool_if:NT \l_@@_width_used_bool
2128 {
2129   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2130   { \@@_error_or_warning:n { width-without-X-columns } }
2131 }

```

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.

```

2132 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2133 \@@_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).

```

2134 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2135 {
2136   \bool_if:NF \l_@@_last_row_without_value_bool
2137   {

```

```

2138         \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2139         {
2140             \@@_error:n { Wrong~last~row }
2141             \int_set_eq:NN \l_@@_last_row_int \c@iRow
2142         }
2143     }
2144 }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` changes: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.<sup>9</sup>

```

2145     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2146     \bool_if:NTF \g_@@_last_col_found_bool
2147     { \int_gdecr:N \c@jCol }
2148     {
2149         \int_compare:nNnT \l_@@_last_col_int > { -1 }
2150         { \@@_error:n { last~col~not~used } }
2151     }

```

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

```

2152     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2153     \int_compare:nNnT \l_@@_last_row_int > { -1 }
2154     { \int_gdecr:N \c@iRow }

```

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

```

2155     \int_if_zero:nT \l_@@_first_col_int
2156     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2157     \bool_if:nTF { ! \g_@@_delims_bool }
2158     {
2159         \str_if_eq:eeTF c \l_@@_baseline_tl
2160         \@@_use_arraybox_with_notes_c:
2161         {
2162             \str_if_eq:eeTF b \l_@@_baseline_tl
2163             \@@_use_arraybox_with_notes_b:
2164             \@@_use_arraybox_with_notes:
2165         }
2166     }

```

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).

```

2167     {
2168         \int_if_zero:nTF \l_@@_first_row_int
2169         {
2170             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2171             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2172         }
2173         { \dim_zero:N \l_tmpa_dim }

```

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

```

2174     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2175     {
2176         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2177         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2178     }
2179     { \dim_zero:N \l_tmpb_dim }
2180     \hbox_set:Nn \l_tmpa_box

```

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

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

```

2181 {
2182   \m@th
2183   $ % $
2184   \@@_color:o \l_@@_delimiters_color_tl
2185   \exp_after:wN \left \g_@@_left_delim_tl
2186   \vcenter
2187   {

```

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.

```

2188     \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2189     \hbox
2190     {
2191       \bool_if:NTF \l_@@_tabular_bool
2192         { \skip_horizontal:n { - \tabcolsep } }
2193         { \skip_horizontal:n { - \arraycolsep } }
2194       \@@_use_arraybox_with_notes_c:
2195       \bool_if:NTF \l_@@_tabular_bool
2196         { \skip_horizontal:n { - \tabcolsep } }
2197         { \skip_horizontal:n { - \arraycolsep } }
2198     }

```

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

```

2199     \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2200   }
2201   \exp_after:wN \right \g_@@_right_delim_tl
2202   $ % $
2203 }

```

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.

```

2204   \bool_if:NTF \l_@@_delimiters_max_width_bool
2205     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2206     \@@_put_box_in_flow:
2207 }

```

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

```

2208   \bool_if:NT \g_@@_last_col_found_bool
2209     { \skip_horizontal:N \g_@@_width_last_col_dim }
2210   \bool_if:NT \l_@@_preamble_bool
2211   {
2212     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2213       { \@@_err_columns_not_used: }
2214   }
2215   \@@_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 exponent to a matrix in a mathematical formula.

```

2216   \egroup

```

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

```

2217   \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2218   \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2219   \iow_now:Ne \@mainaux
2220   {
2221     \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2222     \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2223     { \exp_not:o \g_@@_aux_tl }
2224   }
2225   \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2226   \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2227 }

```

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

```

2228 \cs_new_protected:Npn \@@_err_columns_not_used:
2229 {
2230   \@@_warning:n { columns~not~used }
2231   \cs_gset:Npn \@@_err_columns_not_used: { }
2232 }

```

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  $x$ .

```

2233 \cs_new_protected:Npn \@@_compute_width_X:
2234 {
2235   \tl_gput_right:Ne \g_@@_aux_tl
2236   {
2237     \bool_set_true:N \l_@@_X_columns_aux_bool
2238     \dim_set:Nn \l_@@_X_columns_dim
2239   }

```

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

```

2240   \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2241   { \dim_use:N \l_@@_X_columns_dim }
2242   {
2243     \dim_compare:nNnTF
2244     {
2245       \dim_abs:n
2246       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2247     }
2248     <
2249     { 0.001 pt }
2250     { \dim_use:N \l_@@_X_columns_dim }
2251     {
2252       \dim_eval:n
2253       {
2254         \l_@@_X_columns_dim
2255         +
2256         \fp_to_dim:n
2257         {
2258           (
2259             \dim_eval:n
2260             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2261           )
2262           / \fp_use:N \g_@@_total_X_weight_fp
2263         }
2264       }
2265     }
2266   }
2267 }
2268 }
2269 }

```

## 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`.

```

2270 \cs_new_protected:Npn \@@_transform_preamble:
2271 {
2272   \@@_transform_preamble_i:
2273   \@@_transform_preamble_ii:
2274 }
2275 \cs_new_protected:Npn \@@_transform_preamble_i:
2276 {
2277   \int_gzero:N \c@jCol

```

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

```

2278   \seq_gclear:N \g_@@_cols_vlsim_seq

```

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

```

2279   \bool_gset_false:N \g_tmpb_bool

```

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

```

2280   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2281   \int_zero:N \l_tmpa_int
2282   \tl_gclear:N \g_@@_array_preamble_tl
2283   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2284   {
2285     \tl_gset:Nn \g_@@_array_preamble_tl
2286     { ! { \skip_horizontal:N \arrayrulewidth } }
2287   }
2288   {
2289     \clist_if_in:NnT \l_@@_vlines_clist 1
2290     {
2291       \tl_gset:Nn \g_@@_array_preamble_tl
2292       { ! { \skip_horizontal:N \arrayrulewidth } }
2293     }
2294   }

```

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

```

2295   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2296   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2297   \@@_replace_columncolor:
2298 }

```

```

2299 \cs_new_protected:Npn \@@_transform_preamble_ii:
2300 {

```

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

```

2301   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2302   {
2303     \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2304     { \bool_gset_true:N \g_@@_delims_bool }
2305   }
2306   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2307   \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).

```

2308 \int_if_zero:nTF \l_@@_first_col_int
2309 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2310 {
2311   \bool_if:NF \g_@@_delims_bool
2312   {
2313     \bool_if:NF \l_@@_tabular_bool
2314     {
2315       \clist_if_empty:NT \l_@@_vlines_clist
2316       {
2317         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2318         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2319       }
2320     }
2321   }
2322 }
2323 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2324 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2325 {
2326   \bool_if:NF \g_@@_delims_bool
2327   {
2328     \bool_if:NF \l_@@_tabular_bool
2329     {
2330       \clist_if_empty:NT \l_@@_vlines_clist
2331       {
2332         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2333         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2334       }
2335     }
2336   }
2337 }

```

We try to give a good error message when the final user puts more columns than allowed by the preamble of the array. The mechanism consists of an extra column. However, if tagging is in force, that dummy extra column will be tagged (with <TD> tags) and that’s why we disable that mechanism when tagging is in force.

```

2338 \tag_if_active:F
2339 {

```

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

```

2340 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2341 {
2342   \tl_gput_right:Nn \g_@@_array_preamble_tl
2343   { > { \@@_err_too_many_cols: } 1 }
2344 }
2345 }
2346 }

```

We have used to add a last column to raise a good error message when the user puts more columns than allowed by its preamble. For technical reasons, it was not possible to do that in {NiceTabular\*} and that’s why we used to control that with the value of \l\_@@\_tabular\_width\_dim).

The preamble provided by the final user will be read by a finite automata. The following function \@@\_rec\_preamble:n will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```

2347 \cs_new_protected:Npn \@@_rec_preamble:n #1
2348 {

```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism \csname...\endcsname. Be careful: all



these functions take in as first argument the letter (or token) itself.<sup>11</sup>

```

2349 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2350 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2351 {

```

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

```

2352 \cs_if_exist:cTF { NC @ find @ #1 }
2353 {
2354 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2355 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2356 }
2357 {
2358 \str_if_eq:nnTF { #1 } { S }
2359 { \@@_fatal:n { unknown~column~type~S } }
2360 { \@@_fatal:nn { unknown~column~type } { #1 } }
2361 }
2362 }
2363 }

```

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

```

2364 \cs_new_protected:Npn \@@_c: #1
2365 {
2366 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2367 \tl_gclear:N \g_@@_pre_cell_tl
2368 \tl_gput_right:Nn \g_@@_array_preamble_tl
2369 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2370 \int_gincr:N \c@jCol
2371 \@@_rec_preamble_after_col:n
2372 }

2373 \cs_new_protected:Npn \@@_l: #1
2374 {
2375 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2376 \tl_gclear:N \g_@@_pre_cell_tl
2377 \tl_gput_right:Nn \g_@@_array_preamble_tl
2378 {
2379 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2380 l
2381 < \@@_cell_end:
2382 }
2383 \int_gincr:N \c@jCol
2384 \@@_rec_preamble_after_col:n
2385 }

2386 \cs_new_protected:Npn \@@_r: #1
2387 {
2388 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2389 \tl_gclear:N \g_@@_pre_cell_tl
2390 \tl_gput_right:Nn \g_@@_array_preamble_tl
2391 {
2392 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2393 r
2394 < \@@_cell_end:
2395 }
2396 \int_gincr:N \c@jCol
2397 \@@_rec_preamble_after_col:n
2398 }

```

---

<sup>11</sup>We do that because it's an easy way to insert the letter at some places in the code that we will add to `\g_@@_array_preamble_tl`.

For ! and @

```

2399 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2400 {
2401   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2402   \@@_rec_preamble:n
2403 }
2404 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2405 \cs_new_protected:cpn { @@ _ | : } #1
2406 {
\l_tmpa_int is the number of successive occurrences of |
2407   \int_incr:N \l_tmpa_int
2408   \@@_make_preamble_i_i:n
2409 }
2410 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2411 {

```

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

```

2412   \str_if_eq:nnTF { #1 } { | }
2413   { \use:c { @@ _ | : } | }
2414   { \@@_make_preamble_i_ii:nn { } #1 }
2415 }

```

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

```

2416 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2417 {
2418   \str_if_eq:nnTF { #2 } { [ ]
2419   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2420   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2421   }
2422 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2423 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2424 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2425 {
2426   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2427   \tl_gput_right:Ne \g_@@_array_preamble_tl
2428   {

```

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

```

2429   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2430   }
2431   \tl_gput_right:Ne \g_@@_rules_tl
2432   {

```

With the keys of nicematrix / rules-after we would write:

```

\@@_draw_vrule:n
{
  multiplicity = \int_use:N \l_tmpa_int ,
  position = \int_eval:n { \c@jCol + 1 } ,
  total-width = \dim_use:N \l_@@_rule_width_before_dim ,
  #2
}

```

We will use a version slightly more efficient:

```

2433 {
2434   \int_compare:nNnT \l_tmpa_int > 1
2435   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2436   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2437   \dim_set:Nn \l_@@_rule_width_after_dim

```

```

2438         { \dim_use:N \l_@@_rule_width_before_dim }
2439     \@@_draw_vrule:n { #2 }
2440 }

```

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.

```

2441     }
2442     \int_zero:N \l_tmpa_int
2443     \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2444     \@@_rec_preamble:n #1
2445 }

```

```

2446 \cs_new_protected:cpn { @@_ > : } #1 #2
2447 {
2448     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2449     \@@_rec_preamble:n
2450 }
2451 \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.

```

2452 \keys_define:nn { nicematrix / p-column }
2453 {
2454     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2455     r .value_forbidden:n = true ,
2456     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2457     c .value_forbidden:n = true ,
2458     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2459     l .value_forbidden:n = true ,
2460     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2461     S .value_forbidden:n = true ,
2462     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2463     p .value_forbidden:n = true ,
2464     t .meta:n = p ,
2465     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2466     m .value_forbidden:n = true ,
2467     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2468     b .value_forbidden:n = true
2469 }

```

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

```

2470 \cs_new_protected:Npn \@@_p: #1
2471 {
2472     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2473     \@@_make_preamble_ii_i:n
2474 }
2475 \cs_new_eq:NN \@@_b: \@@_p:
2476 \cs_new_eq:NN \@@_m: \@@_p:
2477 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2478 {
2479     \str_if_eq:nnTF { #1 } { [ ]
2480         { \@@_make_preamble_ii_ii:w [ ]
2481             { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2482         }
2483     \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2484     { \@@_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.

```
2485 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2486 {
```

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).

```
2487 \str_set:Nn \l_@@_hpos_col_str { j }
2488 \@@_keys_p_column:n { #1 }
```

We apply `\setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```
2489 \setlength { \l_tmpa_dim } { #2 }
2490 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2491 }
2492 \cs_new_protected:Npn \@@_keys_p_column:n #1
2493 { \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.

```
2494 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2495 {
```

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

```
2496 \use:e
2497 {
2498 \@@_make_preamble_ii_vi:nnnnnnnn
2499 { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2500 { #1 }
2501 {
2502 \cs_set_eq:NN \rotate \@@_rotate_p_col:
```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_tl` which will provide the horizontal alignment of the column to which belongs the cell.

```
2503 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2504 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2505 {
```

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

```
2506 \def \exp_not:N \l_@@_hpos_cell_tl
2507 { \str_lowercase:f { \l_@@_hpos_col_str } }
2508 }
2509 \IfPackageLoadedTF { ragged2e }
2510 {
2511 \str_case:on \l_@@_hpos_col_str
2512 {
```

The following `\exp_not:N` are mandatory.

```
2513 c { \exp_not:N \Centering }
2514 l { \exp_not:N \RaggedRight }
2515 r { \exp_not:N \RaggedLeft }
2516 }
2517 }
2518 {
2519 \str_case:on \l_@@_hpos_col_str
2520 {
2521 c { \exp_not:N \centering }
2522 l { \exp_not:N \raggedright }
2523 r { \exp_not:N \raggedleft }
```

```

2524     }
2525   }
2526   #3
2527 }
2528 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2529 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2530 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2531 { #2 }
2532 {
2533   \str_case:onF \l_@@_hpos_col_str
2534   {
2535     { j } { c }
2536     { si } { c }
2537   }

```

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

```

2538     { \str_lowercase:f \l_@@_hpos_col_str }
2539   }
2540 }

```

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

```

2541   \int_gincr:N \c@jCol
2542   \@@_rec_preamble_after_col:n
2543 }

```

**#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*.

```

2544 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2545 {
2546   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2547   {
2548     \tl_gput_right:Nn \g_@@_array_preamble_tl
2549     { > \@@_test_if_empty_for_S: }
2550   }
2551   {
2552     \str_if_eq:eeTF { #7 } { varwidth }
2553     {
2554       \tl_gput_right:Nn \g_@@_array_preamble_tl
2555       { > \@@_test_if_empty_varwidth: }
2556     }
2557     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2558   }
2559   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2560   \tl_gclear:N \g_@@_pre_cell_tl
2561   \tl_gput_right:Nn \g_@@_array_preamble_tl
2562   {
2563     > {

```

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.

```

2564     \dim_set:Nn \l_@@_col_width_dim { #2 }
2565     \@@_cell_begin:

```

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

```
2566 \use:c { #7 } [ #1 ] { #2 }
```

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

```
2567 \everypar
2568 {
2569 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2570 \everypar { }
2571 }
```

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

```
2572 #3
```

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

```
2573 \g_@@_row_style_tl
2574 \arraybackslash
2575 #5
2576 }
2577 #8
2578 < {
2579 #6
```

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

```
2580 \@finalstrut \@arstrutbox
2581 \use:c { end #7 }
```

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

```
2582 #4
2583 \@@_cell_end:
2584 }
2585 }
2586 }
```

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

```
2587 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2588 {
```

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 `&`).

```
2589 \group_align_safe_begin:
2590 \peek_meaning:NTF &
2591 \@@_the_cell_is_empty:
2592 {
2593 \peek_meaning:NTF \\\
2594 \@@_the_cell_is_empty:
2595 {
2596 \peek_meaning:NTF \crcr
2597 \@@_the_cell_is_empty:
2598 \group_align_safe_end:
2599 }
2600 }
2601 }
```

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

```
2602 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2603 {
2604 \group_align_safe_begin:
2605 \peek_meaning:NTF &
2606 \@@_the_cell_is_empty_varwidth:
2607 {
```

```

2608     \peek_meaning:NTF \\\
2609     \@@_the_cell_is_empty_varwidth:
2610     {
2611         \peek_meaning:NTF \crcr
2612         \@@_the_cell_is_empty_varwidth:
2613         \group_align_safe_end:
2614     }
2615 }
2616 }

2617 \cs_new_protected:Npn \@@_the_cell_is_empty:
2618 {
2619     \group_align_safe_end:
2620     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2621     {

```

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

```

2622     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```

If all the cells of the column are empty, we still must have a column with the width required by the column of type p (or b, or m).

```

2623     \skip_horizontal:N \l_@@_col_width_dim
2624 }
2625 }

2626 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2627 {
2628     \group_align_safe_end:
2629     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2630     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2631 }

2632 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2633 {
2634     \peek_meaning:NT \__siunitx_table_skip:n
2635     { \bool_gset_true:N \g_@@_empty_cell_bool }
2636 }

```

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.

```

2637 \cs_new_protected:Npn \@@_center_cell_box:
2638 {

```

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

```

2639     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2640     {
2641         \dim_compare:nNnT
2642         { \box_ht:N \l_@@_cell_box }
2643         >

```

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).

```

2644     { \box_ht:N \strutbox }
2645     {
2646         \hbox_set:Nn \l_@@_cell_box
2647         {
2648             \box_move_down:nn
2649             {
2650                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2651             + \baselineskip ) / 2
2652         }
2653         { \box_use:N \l_@@_cell_box }
2654     }
2655 }
2656 }
2657 }

```

For V (similar to the V of varwidth).

```

2658 \cs_new_protected:Npn \@@_V: #1 #2
2659 {
2660     \str_if_eq:nnTF { #2 } { [ ] }
2661     { \@@_make_preamble_V_i:w [ ] }
2662     { \@@_make_preamble_V_i:w [ ] { #2 } }
2663 }
2664 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2665 { \@@_make_preamble_V_ii:nn { #1 } }
2666 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2667 {
2668     \str_set:Nn \l_@@_vpos_col_str { p }
2669     \str_set:Nn \l_@@_hpos_col_str { j }
2670     \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2671     \setlength { \l_tmpa_dim } { #2 }
2672     \IfPackageLoadedTF { varwidth }
2673     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2674     {
2675         \@@_error_or_warning:n { varwidth~not~loaded }
2676         \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2677     }
2678 }
2679 % \end{macrocod}
2680 %
2681 % \medskip
2682 % For |w| and |W|
2683 % \begin{macrocode}
2684 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2685 \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. It is provided by curryfication.

```

2686 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2687 {
2688     \tl_if_in:nnTF { clr } { #3 }
2689     { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2690     {
2691         \@@_error:nn { Invalid~argument~for~w } { #3 }
2692         \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2693     }
2694 }
2695 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2696 {
2697     \str_if_eq:nnTF { #3 } { s }
2698     { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2699     { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2700 }

```



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.

```

2701 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2
2702 {
2703   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2704   \tl_gclear:N \g_@@_pre_cell_tl
2705   \tl_gput_right:Nn \g_@@_array_preamble_tl
2706   {
2707     > {

```

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2708       \setlength { \l_@@_col_width_dim } { #2 }
2709       \@@_cell_begin:
2710       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2711     }
2712   c
2713   < {
2714     \@@_cell_end_for_w_s:
2715     #1
2716     \@@_adjust_size_box:
2717     \box_use_drop:N \l_@@_cell_box
2718   }
2719 }
2720 \int_gincr:N \c@jCol
2721 \@@_rec_preamble_after_col:n
2722 }

```

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

```

2723 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2724 {
2725   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2726   \tl_gclear:N \g_@@_pre_cell_tl
2727   \tl_gput_right:Nn \g_@@_array_preamble_tl
2728   {
2729     > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2730       \setlength { \l_@@_col_width_dim } { #4 }
2731       \hbox_set:Nw \l_@@_cell_box
2732       \@@_cell_begin:
2733       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2734     }
2735   c
2736   < {
2737     \@@_cell_end:
2738     \hbox_set_end:
2739     #1
2740     \@@_adjust_size_box:
2741     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2742   }
2743 }

```

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

```

2744   \int_gincr:N \c@jCol
2745   \@@_rec_preamble_after_col:n
2746 }

```

```

2747 \cs_new_protected:Npn \@@_special_W:
2748 {
2749   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2750   { \@@_warning:n { W~warning } }
2751 }

```

For S (of siunitx).

```

2752 \AtBeginDocument
2753 {
2754   \IfPackageLoadedT { siunitx }
2755   {
2756     \cs_new_protected:Npn \@@_S: #1 #2
2757     {
2758       \str_if_eq:nnTF { #2 } { [ ] }
2759       { \@@_make_preamble_S:w [ ] }
2760       { \@@_make_preamble_S:w [ ] { #2 } }
2761     }
2762   }
2763 }
2764 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2765 { \@@_make_preamble_S_i:n { #1 } }
2766 \cs_new_protected:Npn \@@_test_siunitx:
2767 {
2768   \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2769   { \@@_fatal:n { siunitx~too~old } }
2770   \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2771 }
2772 % \end{macrocode}
2773 %
2774 % \begin{macrocode}
2775 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2776 {
2777   \@@_test_siunitx:
2778   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2779   \tl_gclear:N \g_@@_pre_cell_tl
2780   \tl_gput_right:Nn \g_@@_array_preamble_tl
2781   {
2782     > {

```

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).

```

2783       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2784       \keys_set:nn { siunitx } { #1 }
2785       \@@_cell_begin:
2786       \siunitx_cell_begin:w
2787     }
2788     c
2789     <
2790     {
2791       \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 it will stay local within the cell of the underlying `\halign`).

```

2792       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2793       {
2794         \bool_if:NTF \l__siunitx_table_text_bool
2795         \bool_set_true:N
2796         \bool_set_false:N
2797         \l__siunitx_table_text_bool

```

```

2798         }
2799         \@@_cell_end:
2800     }
2801 }

```

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

```

2802     \int_gincr:N \c@jCol
2803     \@@_rec_preamble_after_col:n
2804 }

```

For (, [ and \{.

```

2805 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2806 {
2807     \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.

```

2808     \int_if_zero:nTF \c@jCol
2809     {
2810         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2811         {

```

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

```

2812             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2813             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2814             \@@_rec_preamble:n #2
2815         }
2816         {
2817             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2818             \@@_make_preamble_iv:nn { #1 } { #2 }
2819         }
2820     }
2821     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2822 }
2823 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2824 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2825 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2826 {
2827     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2828     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2829     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2830     {
2831         \@@_error:nn { delimiter~after~opening } { #2 }
2832         \@@_rec_preamble:n
2833     }
2834     { \@@_rec_preamble:n #2 }
2835 }

```

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

```

2836 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2837 { \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 a 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}).

```

2838 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2839 {
2840     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2841     \tl_if_in:nnTF { ) ] \} } { #2 }
2842     { \@@_make_preamble_v:nnn #1 #2 }
2843     {
2844         \str_if_eq:nnTF \s_stop { #2 }

```

```

2845 {
2846   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2847   { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2848   {
2849     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2850     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2851     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2852     \@@_rec_preamble:n #2
2853   }
2854 }
2855 {
2856   \tl_if_in:nnT { ( [ \{ \left } { #2 }
2857   { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2858   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2859   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2860   \@@_rec_preamble:n #2
2861 }
2862 }
2863 }
2864 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2865 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2866 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2867 {
2868   \str_if_eq:nnTF \s_stop { #3 }
2869   {
2870     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2871     {
2872       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2873       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2874       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2875       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2876     }
2877     {
2878       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2879       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2880       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2881       \@@_error:nn { double~closing~delimiter } { #2 }
2882     }
2883   }
2884   {
2885     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2886     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2887     \@@_error:nn { double~closing~delimiter } { #2 }
2888     \@@_rec_preamble:n #3
2889   }
2890 }
2891 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2892 { \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 @{...}.

```

2893 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2894 {
2895   \str_if_eq:nnTF { #1 } { < }
2896   { \@@_rec_preamble_after_col_i:n }
2897   {
2898     \str_if_eq:nnTF { #1 } { @ }
2899     { \@@_rec_preamble_after_col_ii:n }
2900     {
2901       \str_if_eq:eeTF \l_@@_vlines_clist { all }

```

```

2902     {
2903       \tl_gput_right:Nn \g_@@_array_preamble_tl
2904       { ! { \skip_horizontal:N \arrayrulewidth } }
2905     }
2906     {
2907       \clist_if_in:NcT \l_@@_vlines_clist
2908       { \int_eval:n { \c@jCol + 1 } }
2909       {
2910         \tl_gput_right:Nn \g_@@_array_preamble_tl
2911         { ! { \skip_horizontal:N \arrayrulewidth } }
2912       }
2913     }
2914     \@@_rec_preamble:n { #1 }
2915   }
2916 }
2917 }
2918 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2919 {
2920   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2921   \@@_rec_preamble_after_col:n
2922 }

```

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.

```

2923 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2924 {
2925   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2926   {
2927     \tl_gput_right:Nn \g_@@_array_preamble_tl
2928     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2929   }
2930   {
2931     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2932     {
2933       \tl_gput_right:Nn \g_@@_array_preamble_tl
2934       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2935     }
2936     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2937   }
2938   \@@_rec_preamble:n
2939 }
2940 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2941 {
2942   \tl_clear:N \l_tmpa_tl
2943   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2944   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2945 }

```

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

```

2946 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2947 { \@@_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.

```

2948 \cs_new_protected:Npn \@@_X: #1 #2
2949 {
2950   \str_if_eq:nnTF { #2 } { [ ]
2951     { \@@_make_preamble_X:w [ ]
2952       { \@@_make_preamble_X:w [ ] #2 }
2953     }

```

```

2954 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2955 { \@@_make_preamble_X_i:n { #1 } }

```

#1 is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of { `nicematrix / p-column` } but also a key V and also a key which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```

2956 \keys_define:nn { nicematrix / X-column }
2957 {
2958   V .code:n =
2959     \IfPackageLoadedTF { varwidth }
2960     {
2961       \bool_set_true:N \l_@@_V_of_X_bool
2962       \bool_gset_true:N \g_@@_V_of_X_bool
2963     }
2964     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2965   unknown .code:n =
2966     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2967     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2968     { \@@_error_or_warning:n { invalid~weight } }
2969 }

```

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

```

2970 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2971 {

```

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).

```

2972   \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).

```

2973   \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`).

```

2974   \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

2975   \@@_keys_p_column:n { #1 }

```

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

```

2976   \bool_set_false:N \l_@@_V_of_X_bool
2977   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2978   \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).

```

2979   \bool_if:NTF \l_@@_X_columns_aux_bool
2980   {
2981     \@@_make_preamble_ii_iv:nnn

```

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

```

2982     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2983     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2984     { \@@_no_update_width: }
2985   }

```

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.

```

2986 {
2987   \tl_gput_right:Nn \g_@@_array_preamble_tl
2988   {
2989     > {
2990       \@@_cell_begin:
2991       \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`.

```

2992       \NotEmpty

```

The following code will nullify the box of the cell.

```

2993       \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2994       { \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`.

```

2995       \begin { minipage } { 5 cm } \arraybackslash
2996     }
2997     c
2998     < {
2999       \end { minipage }
3000       \@@_cell_end:
3001     }
3002   }
3003   \int_gincr:N \c@jCol
3004   \@@_rec_preamble_after_col:n
3005 }
3006 }

3007 \cs_new_protected:Npn \@@_no_update_width:
3008 {
3009   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3010   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3011 }

```

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

```

3012 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3013 {
3014   \seq_gput_right:Ne \g_@@_cols_vlism_seq
3015   { \int_eval:n { \c@jCol + 1 } }
3016   \tl_gput_right:Ne \g_@@_array_preamble_tl
3017   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3018   \@@_rec_preamble:n
3019 }

```

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.

```

3020 \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, for example, forgotten the preamble of its environment).

```

3021 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3022 { \@@_fatal:n { Preamble-forgotten } }
3023 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3024 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3025 { @@ _ \token_to_str:N \hline : }
3026 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }

```

```

3027 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3028 { @@ _ \token_to_str:N \hline : }
3029 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3030 { @@ _ \token_to_str:N \hline : }
3031 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3032 { @@ _ \token_to_str:N \hline : }
3033 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3034 { @@ _ \token_to_str:N \hline : }
3035 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3036 { @@_fatal:n { NiceTabularX~probably~required } }
3037 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3038 { @@ _ \token_to_str:N \linewidth : }

```

## 12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

3039 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3040 {

```

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`.

```

3041 \multispan { #1 }
3042 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3043 \begingroup
3044 \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3045 \tl_gclear:N \g_@@_preamble_tl
3046 \@@_make_m_preamble:n #2 \q_stop

```

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

```

3047 \def \@addamp
3048 {
3049 \legacy_if:nTF { @firstamp }
3050 { \legacy_if_set_false:n { @firstamp } }
3051 { \@preamerr 5 }
3052 }
3053 \exp_args:No \@mkpream \g_@@_preamble_tl
3054 \@addtopreamble \empty
3055 \endgroup
3056 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3057 \int_compare:nNnT { #1 } > 1
3058 {
3059 \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3060 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3061 \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3062 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3063 {
3064 {
3065 \int_if_zero:nTF \c@jCol
3066 { \int_eval:n { \c@iRow + 1 } }
3067 { \int_use:N \c@iRow }
3068 }
3069 { \int_eval:n { \c@jCol + 1 } }

```



```

3070      {
3071        \int_if_zero:nTF \c@jCol
3072        { \int_eval:n { \c@iRow + 1 } }
3073        { \int_use:N \c@iRow }
3074      }
3075      { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3076      { }
3077    }
3078  }

```

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

```

3079  \RenewDocumentCommand { \cellcolor } { 0 { } m }
3080  {
3081    \tl_gput_right:Ne \g_@@_pre_code_before_tl
3082    {
3083      \@@_rectanglecolor [ ##1 ]
3084      { \exp_not:n { ##2 } }
3085      { \int_use:N \c@iRow - \int_use:N \c@jCol }
3086      { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3087    }
3088    \ignorespaces
3089  }

```

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

```

3090  \def \@sharp { #3 }
3091  \@arstrut
3092  \@preamble
3093  \null

```

We add some lines.

```

3094  \int_gadd:Nn \c@jCol { #1 - 1 }
3095  \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3096  { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3097  \ignorespaces
3098  }

```

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`.

```

3099  \
3100  \cs_new_protected:Npn \@_make_m_preamble:n #1
3101  {
3102    \str_case:nnF { #1 }
3103    {
3104      c { \@_make_m_preamble_i:n #1 }
3105      l { \@_make_m_preamble_i:n #1 }
3106      X { \@_make_m_preamble_i:n l }
3107      r { \@_make_m_preamble_i:n #1 }
3108      > { \@_make_m_preamble_ii:nn #1 }
3109      ! { \@_make_m_preamble_ii:nn #1 }
3110      @ { \@_make_m_preamble_ii:nn #1 }
3111      | { \@_make_m_preamble_iii:n #1 }
3112      p { \@_make_m_preamble_iv:nnn t #1 }
3113      m { \@_make_m_preamble_iv:nnn c #1 }
3114      b { \@_make_m_preamble_iv:nnn b #1 }
3115      w { \@_make_m_preamble_v:nnnn { } #1 }
3116      W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
3117      \q_stop { }
3118    }
3119  {

```

```

3120 \cs_if_exist:cTF { NC @ find @ #1 }
3121 {
3122   \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3123   \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3124 }
3125 {
3126   \str_if_eq:nnTF { #1 } { S }
3127   { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3128   { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3129 }
3130 }
3131 }

```

For c, l and r

```

3132 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3133 {
3134   \tl_gput_right:Nn \g_@@_preamble_tl
3135   {
3136     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3137     #1
3138     < \@@_cell_end:
3139   }

```

We test for the presence of a <.

```

3140   \@@_make_m_preamble_x:n
3141 }

```

For >, ! and @

```

3142 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3143 {
3144   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3145   \@@_make_m_preamble:n
3146 }

```

For |

```

3147 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3148 {
3149   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3150   \@@_make_m_preamble:n
3151 }

```

For p, m and b

```

3152 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3153 {
3154   \tl_gput_right:Nn \g_@@_preamble_tl
3155   {
3156     > {
3157       \@@_cell_begin:

```

We use `\setlength` instead of `\dim_set:N` to allow a specifier like `p{\widthof{Some words}}`. `widthof` is a command provided by `calc`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

3158   \setlength { \l_tmpa_dim } { #3 }
3159   \begin { minipage } [ #1 ] { \l_tmpa_dim }
3160   \mode_leave_vertical:
3161   \arraybackslash
3162   \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3163   }
3164   c
3165   < {
3166     \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3167     \end { minipage }
3168     \@@_cell_end:

```

```

3169     }
3170 }

```

We test for the presence of a <.

```

3171 \@@_make_m_preamble_x:n
3172 }

```

For w and W

```

3173 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3174 {
3175   \tl_gput_right:Nn \g_@@_preamble_tl
3176   {
3177     > {
3178       \dim_set:Nn \l_@@_col_width_dim { #4 }
3179       \hbox_set:Nw \l_@@_cell_box
3180       \@@_cell_begin:
3181       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3182     }
3183     c
3184     < {
3185       \@@_cell_end:
3186       \hbox_set_end:
3187       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3188       #1
3189       \@@_adjust_size_box:
3190       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3191     }
3192   }

```

We test for the presence of a <.

```

3193 \@@_make_m_preamble_x:n
3194 }

```

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

```

3195 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3196 {
3197   \str_if_eq:nnTF { #1 } { < }
3198   \@@_make_m_preamble_ix:n
3199   { \@@_make_m_preamble:n { #1 } }
3200 }
3201 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3202 {
3203   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3204   \@@_make_m_preamble_x:n
3205 }

```

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).

```

3206 \cs_new_protected:Npn \@@_put_box_in_flow:
3207 {
3208   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3209   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3210   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3211   { \box_use_drop:N \l_tmpa_box }
3212   \@@_put_box_in_flow_i:
3213 }

```

The command \@@\_put\_box\_in\_flow\_i: is used when the value of \l\_@@\_baseline\_tl is different of c (the initial value).

```

3214 \cs_new_protected:Npn \@@_put_box_in_flow_i:

```

```

3215 {
3216   \pgfpicture
3217     \@@_qpoint:n { row - 1 }
3218     \dim_gset:nn \g_tmpa_dim \pgf@y
3219     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3220     \dim_gadd:nn \g_tmpa_dim \pgf@y
3221     \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).

```

3222   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3223   {
3224     \int_set:nn \l_tmpa_int
3225     { \str_range:nnn \l_@@_baseline_tl { 6 } { -1 } }
3226     \bool_lazy_or:nnT
3227     { \int_compare_p:nnn \l_tmpa_int < { 1 } }
3228     { \int_compare_p:nnn \l_tmpa_int > { \c@iRow + 1 } }
3229     {
3230       \@@_error:n { bad-value-for-baseline-line }
3231       \int_set:nn \l_tmpa_int 1
3232     }
3233     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3234   }
3235   {
3236     \str_if_eq:eeTF t \l_@@_baseline_tl
3237     { \int_set:nn \l_tmpa_int 1 }
3238     {
3239       \str_if_eq:eeTF b \l_@@_baseline_tl
3240       { \int_set_eq:nn \l_tmpa_int \c@iRow }
3241       { \int_set:nn \l_tmpa_int \l_@@_baseline_tl }
3242     }
3243     \bool_lazy_or:nnT
3244     { \int_compare_p:nnn \l_tmpa_int < \l_@@_first_row_int }
3245     { \int_compare_p:nnn \l_tmpa_int > \g_@@_row_total_int }
3246     {
3247       \@@_error:n { bad-value-for-baseline }
3248       \int_set:nn \l_tmpa_int 1
3249     }
3250     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3251     \dim_gsub:nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3252   }
3253   \dim_gsub:nn \g_tmpa_dim \pgf@y

```

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

```

3254   \endpgfpicture
3255   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3256   % \box_use_drop:N \l_tmpa_box   % 2026/04/13
3257 }

```

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).

```

3258 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3259 {

```

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.

```

3260   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3261   {
3262     \int_compare:nnnT \c@jCol > 1
3263     {

```

```

3264         \box_set_wd:Nn \l_@@_the_array_box
3265         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3266     }
3267 }

```

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

```

3268 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3269 \bool_if:NT \l_@@_caption_above_bool
3270 {
3271     \tl_if_empty:NF \l_@@_caption_tl
3272     {
3273         \bool_set_false:N \g_@@_caption_finished_bool
3274         \int_gzero:N \c@tabularnote
3275         \@@_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).

```

3276         \int_compare:nNtT \g_@@_notes_caption_int > \c_zero_int
3277         {
3278             \tl_gput_right:Ne \g_@@_aux_tl
3279             {
3280                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3281                 { \int_use:N \g_@@_notes_caption_int }
3282             }
3283             \int_gzero:N \g_@@_notes_caption_int
3284         }
3285     }
3286 }

```

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

```

3287 \hbox
3288 {
3289     \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right away because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are `medium` nodes to create for the blocks.

```

3290 \@@_create_extra_nodes:
3291 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3292 }

```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because it compiles twice its tabular).

```

3293 \bool_lazy_any:nT
3294 {
3295     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3296     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3297     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3298 }
3299 {
3300     \bool_if:NTF \l_@@_notes_no_print_bool
3301     { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3302     \@@_tabular_notes:
3303 }
3304 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3305 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3306 \end { minipage }
3307 }

```

```

3308 \cs_new_protected:Npn \@@_insert_caption:
3309 {
3310   \tl_if_empty:NF \l_@@_caption_tl
3311   {
3312     \cs_if_exist:NTF \@capttype
3313     { \@@_insert_caption_i: }
3314     { \@@_error:n { caption-outside-float } }
3315   }
3316 }

```

```

3317 \cs_new_protected:Npn \@@_insert_caption_i:
3318 {
3319   \group_begin:

```

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

```

3320   \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.

```

3321   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3322   \tl_if_empty:NTF \l_@@_short_caption_tl
3323   \caption
3324   { \caption [ \l_@@_short_caption_tl ] }
3325   { \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.

```

3326   \bool_if:NF \g_@@_caption_finished_bool
3327   {
3328     \bool_gset_true:N \g_@@_caption_finished_bool
3329     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3330     \int_gzero:N \c@tabularnote
3331   }
3332   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3333   \group_end:
3334 }

```

```

3335 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3336 {
3337   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3338   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3339 }

```

```

3340 \cs_set_protected:Npn \@@_tabular_notes_error:
3341 { \@@_error:n { Bad-use-of-NiceTabularNotes } }
3342 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3343 \cs_set_protected:Npn \@@_tabular_notes:
3344 {
3345   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3346   \seq_concat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3347   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3348   \skip_vertical:N 0.65ex

```

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

```

3349   \group_begin:
3350   \l_@@_notes_code_before_tl
3351   \tl_if_empty:NF \g_@@_tabularnote_tl
3352   {
3353     \g_@@_tabularnote_tl \par

```

```

3354     \tl_gclear:N \g_@@_tabularnote_tl
3355 }

```

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.

```

3356 \int_compare:nNnT \c@tabularnote > \c_zero_int
3357 {
3358   \bool_if:NTF \l_@@_notes_para_bool
3359   {
3360     \begin { tabularnotes* }
3361     \seq_map_inline:Nn \g_@@_notes_seq
3362       { \@@_one_tabularnote:nn ##1 }
3363     \strut
3364     \end { tabularnotes* }

```

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

```

3365     \par
3366   }
3367 {
3368   \tabularnotes
3369   \seq_map_inline:Nn \g_@@_notes_seq
3370     { \@@_one_tabularnote:nn ##1 }
3371   \strut
3372   \endtabularnotes
3373 }
3374 }
3375 \unskip
3376 \group_end:
3377 \bool_if:NNT \l_@@_notes_bottomrule_bool
3378 {
3379   \IfPackageLoadedTF { booktabs }
3380   {

```

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

```

3381     \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
3382     { \CT@arc@ \hrule height \heavyrulewidth }
3383   }
3384   { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3385 }
3386 \l_@@_notes_code_after_tl
3387 \seq_gclear:N \g_@@_notes_seq
3388 \seq_gclear:N \g_@@_notes_in_caption_seq
3389 \int_gzero:N \c@tabularnote
3390 }

```

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.

```

3391 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3392 {
3393   \tl_if_novalue:nTF { #1 }
3394   { \item }
3395   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3396 }

```

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`.

```

3397 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3398 {

```

```

3399 \pgfpicture
3400 \@@_qpoint:n { row - 1 }
3401 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3402 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3403 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3404 \endpgfpicture
3405 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3406 \int_if_zero:nT \l_@@_first_row_int
3407 {
3408 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3409 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3410 }
3411 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3412 }

```

Now, the general case.

```

3413 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3414 {

```

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

```

3415 \str_if_eq:eeT \l_@@_baseline_tl { t }
3416 { \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`.

```

3417 \pgfpicture
3418 \@@_qpoint:n { row - 1 }
3419 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3420 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3421 {
3422 \int_set:Nn \l_tmpa_int
3423 {
3424 \str_range:Nnn
3425 \l_@@_baseline_tl
3426 { 6 }
3427 { \tl_count:o \l_@@_baseline_tl }
3428 }
3429 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3430 }
3431 {
3432 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3433 \bool_lazy_or:nnT
3434 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3435 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3436 {
3437 \@@_error:n { bad~value~for~baseline }
3438 \int_set:Nn \l_tmpa_int 1
3439 }
3440 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3441 }
3442 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3443 \endpgfpicture
3444 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3445 \int_if_zero:nT \l_@@_first_row_int
3446 {
3447 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3448 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3449 }
3450 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3451 }

```

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.



```

3452 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3453 {

```

We will compute the real width of both delimiters used.

```

3454 \dim_zero_new:N \l_@@_real_left_delim_dim
3455 \dim_zero_new:N \l_@@_real_right_delim_dim
3456 \hbox_set:Nn \l_tmpb_box
3457 {
3458   \m@th
3459   $ % $
3460   \left #1
3461   \vcenter
3462   {
3463     \vbox_to_ht:nn
3464     { \box_ht_plus_dp:N \l_tmpa_box }
3465     { }
3466   }
3467   \right .
3468   $ % $
3469 }
3470 \dim_set:Nn \l_@@_real_left_delim_dim
3471 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3472 \hbox_set:Nn \l_tmpb_box
3473 {
3474   \m@th
3475   $ % $
3476   \left .
3477   \vbox_to_ht:nn
3478   { \box_ht_plus_dp:N \l_tmpa_box }
3479   { }
3480   \right #2
3481   $ % $
3482 }
3483 \dim_set:Nn \l_@@_real_right_delim_dim
3484 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3485 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3486 \@@_put_box_in_flow:
3487 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3488 }

```

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).

```

3489 \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).

```

3490 {
3491   \peek_remove_spaces:n
3492   {
3493     \peek_meaning:NTF \end
3494     \@@_analyze_end:Nn
3495     {
3496       \@@_transform_preamble:
3497       \@@_array:o \g_@@_array_preamble_tl
3498     }
3499   }
3500 }

```

```

3501 {
3502   \@@_create_col_nodes:
3503   \endarray
3504 }

```

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

```

3505 \NewDocumentEnvironment { @@-light-syntax } { b }
3506 {

```

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`.

```

3507   \tl_if_empty:nT { #1 }
3508   { \@@_fatal:n { empty-environment } }
3509   \tl_if_in:nnT { #1 } { & }
3510   { \@@_fatal:n { ampersand-in-light-syntax } }
3511   \tl_if_in:nnT { #1 } { \ }
3512   { \@@_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`.

```

3513   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3514 }

```

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.

```

3515 {
3516   \@@_create_col_nodes:
3517   \endarray
3518 }
3519 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3520 {
3521   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

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

```

3522   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3523   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3524   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3525   \seq_set_split:Nee
3526   \seq_set_split:Non
3527   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3528   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3529   \tl_if_empty:NF \l_tmpa_tl
3530   { \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.

```

3531   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3532   { \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`).

```
3533 \tl_build_begin:N \l_@@_new_body_tl
3534 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3535 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3536 \@@_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).

```
3537 \seq_map_inline:Nn \l_@@_rows_seq
3538 {
3539   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3540   \@@_line_with_light_syntax:n { #1 }
3541 }
3542 \tl_build_end:N \l_@@_new_body_tl
3543 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3544 {
3545   \int_set:Nn \l_@@_last_col_int
3546     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3547 }
```

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.

```
3548 \@@_transform_preamble:

3549 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3550 }
3551 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3552 {
3553   \seq_clear_new:N \l_@@_cells_seq
3554   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3555   \int_set:Nn \l_@@_nb_cols_int
3556     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3557   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3558   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3559   \seq_map_inline:Nn \l_@@_cells_seq
3560     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3561 }
3562 \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`.

```
3563 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3564 {
3565   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3566     { \@@_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.

```
3567 \end { #2 }
3568 }
```

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`).

```
3569 \cs_new:Npn \@@_create_col_nodes:
3570 {
3571   \crrc
```

```

3572 \int_if_zero:nT \l_@@_first_col_int
3573 {
3574   \omit
3575   \hbox_overlap_left:n
3576   {
3577     \bool_if:NT \l_@@_code_before_bool
3578     { \pgfsys@markposition { \@@_env: - col - 0 } }
3579     \pgfpicture
3580     \pgfrememberpicturepositiononpagetrue
3581     \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3582     \str_if_empty:NF \l_@@_name_str
3583     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3584     \endpgfpicture
3585     \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3586   }
3587   &
3588 }
3589 \omit

```

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

```

3590 \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`).

```

3591 \int_if_zero:nTF \l_@@_first_col_int
3592 {
3593   \@@_mark_position:n { 1 }
3594   \pgfpicture
3595   \pgfrememberpicturepositiononpagetrue
3596   \pgfcoordinate { \@@_env: - col - 1 }
3597   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3598   \str_if_empty:NF \l_@@_name_str
3599   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3600   \endpgfpicture
3601 }
3602 {
3603   \bool_if:NT \l_@@_code_before_bool
3604   {
3605     \hbox
3606     {
3607       \skip_horizontal:n { 0.5 \arrayrulewidth }
3608       \pgfsys@markposition { \@@_env: - col - 1 }
3609       \skip_horizontal:n { -0.5 \arrayrulewidth }
3610     }
3611   }
3612   \pgfpicture
3613   \pgfrememberpicturepositiononpagetrue
3614   \pgfcoordinate { \@@_env: - col - 1 }
3615   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3616   \@@_node_alias:n { 1 }
3617   \endpgfpicture
3618 }

```

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.

```

3619 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3620 \bool_if:NF \l_@@_auto_columns_width_bool
3621 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3622 {
3623   \bool_lazy_and:nnTF

```

```

3624 \l_@@_auto_columns_width_bool
3625 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3626 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3627 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3628 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3629 }
3630 \skip_horizontal:N \g_tmpa_skip
3631 \hbox
3632 {
3633 \@@_mark_position:n { 2 }
3634 \pgfpicture
3635 \pgfrememberpicturerepositiononpagetrue
3636 \pgfcoordinate { \@@_env: - col - 2 }
3637 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3638 \@@_node_alias:n { 2 }
3639 \endpgfpicture
3640 }

```

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.

```

3641 \int_gset:Nn \g_tmpa_int 1
3642 \bool_if:NTF \g_@@_last_col_found_bool
3643 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3644 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3645 {
3646 &
3647 \omit
3648 \int_gincr:N \g_tmpa_int

```

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

```

3649 \skip_horizontal:N \g_tmpa_skip
3650 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3651 \pgfpicture
3652 \pgfrememberpicturerepositiononpagetrue
3653 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3654 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3655 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3656 \endpgfpicture
3657 }

```

If there is only one column (and a potential “last column”), we don’t have to put the following code (there is only one column and we have put the correct code previously).

```

3658 \bool_lazy_or:nnF
3659 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3660 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3661 {
3662 &
3663 \omit
3664 \skip_horizontal:N \g_tmpa_skip
3665 \int_gincr:N \g_tmpa_int
3666 \bool_lazy_any:nF
3667 {
3668 \g_@@_delims_bool
3669 \l_@@_tabular_bool
3670 { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3671 \l_@@_exterior_arraycolsep_bool
3672 \l_@@_bar_at_end_of_pream_bool
3673 }
3674 { \skip_horizontal:n { - \col@sep } }
3675 \bool_if:NT \l_@@_code_before_bool
3676 {
3677 \hbox
3678 {

```

```
3679 \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.

```
3680 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3681 { \skip_horizontal:n { - \arraycolsep } }
3682 \pgfsys@markposition
3683 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3684 \skip_horizontal:n { 0.5 \arrayrulewidth }
3685 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3686 { \skip_horizontal:N \arraycolsep }
3687 }
3688 }
3689 \pgfpicture
3690 \pgfrememberpicturepositiononpagetrue
3691 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3692 {
3693 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3694 {
3695 \pgfpoint
3696 { - 0.5 \arrayrulewidth - \arraycolsep }
3697 \c_zero_dim
3698 }
3699 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3700 }
3701 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3702 \endpgfpicture
3703 }
```

```
3704 \bool_if:NT \g_@@_last_col_found_bool
3705 {
3706 \hbox_overlap_right:n
3707 {
3708 \skip_horizontal:N \g_@@_width_last_col_dim
3709 \skip_horizontal:N \col@sep
3710 \bool_if:NT \l_@@_code_before_bool
3711 {
3712 \pgfsys@markposition
3713 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3714 }
3715 \pgfpicture
3716 \pgfrememberpicturepositiononpagetrue
3717 \pgfcoordinate
3718 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3719 \pgfpointorigin
3720 \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3721 \endpgfpicture
3722 }
3723 }
3724 }
```

```
3725 \cs_new_protected:Npn \@@_mark_position:n #1
3726 {
3727 \bool_if:NT \l_@@_code_before_bool
3728 {
3729 \hbox
3730 {
3731 \skip_horizontal:n { -0.5 \arrayrulewidth }
3732 \pgfsys@markposition { \@@_env: - col - #1 }
3733 \skip_horizontal:n { 0.5 \arrayrulewidth }
3734 }
```

```

3735     }
3736 }
3737 \cs_new_protected:Npn \@@_node_alias:n #1
3738 {
3739   \str_if_empty:NF \l_@@_name_str
3740   { \pgfnodelalias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3741 }

```

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

```

3742 \tl_const:Nn \c_@@_preamble_first_col_tl
3743 {
3744   >
3745   {

```

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

```

3746     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3747     \bool_gset_true:N \g_@@_after_col_zero_bool
3748     \@@_begin_of_row:
3749     \hbox_set:Nw \l_@@_cell_box
3750     \@@_math_toggle:
3751     \@@_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”.

```

3752     \int_compare:nNnT \c@iRow > \c_zero_int
3753     {
3754       \bool_lazy_or:nnT
3755       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3756       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3757       {
3758         \l_@@_code_for_first_col_tl
3759         \xglobal \colorlet { nicematrix-first-col } { . }
3760       }
3761     }
3762 }

```

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`.

```

3763   l
3764   <
3765   {
3766     \@@_math_toggle:
3767     \hbox_set_end:
3768     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3769     \@@_adjust_size_box:
3770     \@@_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.

```

3771     \dim_gset:Nn \g_@@_width_first_col_dim
3772     { \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.

```

3773     \hbox_overlap_left:n
3774     {
3775       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3776       \@@_node_cell:
3777       { \box_use_drop:N \l_@@_cell_box }
3778       \skip_horizontal:N \l_@@_left_delim_dim
3779       \skip_horizontal:N \l_@@_left_margin_dim
3780       \skip_horizontal:N \l_@@_extra_left_margin_dim
3781     }

```

```

3782         \bool_gset_false:N \g_@@_empty_cell_bool
3783         \skip_horizontal:n { -2 \col@sep }
3784     }
3785 }

```

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

```

3786 \tl_const:Nn \c_@@_preamble_last_col_tl
3787 {
3788     >
3789     {
3790         \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).

```

3791         \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.

```

3792         \bool_gset_true:N \g_@@_last_col_found_bool
3793         \int_gincr:N \c@jCol
3794         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3795         \hbox_set:Nw \l_@@_cell_box
3796         \@@_math_toggle:
3797         \@@_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”.

```

3798         \int_compare:nNnT \c@iRow > \c_zero_int
3799         {
3800             \bool_lazy_or:nnT
3801             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3802             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3803             {
3804                 \l_@@_code_for_last_col_tl
3805                 \xglobal \colorlet { nicematrix-last-col } { . }
3806             }
3807         }
3808     }
3809     l
3810     <
3811     {
3812         \@@_math_toggle:
3813         \hbox_set_end:
3814         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3815         \@@_adjust_size_box:
3816         \@@_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.

```

3817         \dim_gset:Nn \g_@@_width_last_col_dim
3818         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3819         \skip_horizontal:n { -2 \col@sep }

```

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

```

3820         \hbox_overlap_right:n
3821         {
3822             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3823             {
3824                 \skip_horizontal:N \l_@@_right_delim_dim
3825                 \skip_horizontal:N \l_@@_right_margin_dim
3826                 \skip_horizontal:N \l_@@_extra_right_margin_dim
3827                 \@@_node_cell:
3828             }
3829         }
3830         \bool_gset_false:N \g_@@_empty_cell_bool
3831     }
3832 }

```



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

```

3833 \NewDocumentEnvironment { NiceArray } { }
3834 {
3835   \bool_gset_false:N \g_@@_delims_bool
3836   \str_if_empty:NT \g_@@_name_env_str
3837   { \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).

```

3838   \NiceArrayWithDelims . .
3839 }
3840 { \endNiceArrayWithDelims }

```

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

```

3841 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3842 {
3843   \NewDocumentEnvironment { #1 NiceArray } { }
3844   {
3845     \bool_gset_true:N \g_@@_delims_bool
3846     \str_if_empty:NT \g_@@_name_env_str
3847     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3848     \@@_test_if_math_mode:
3849     \NiceArrayWithDelims #2 #3
3850   }
3851   { \endNiceArrayWithDelims }
3852 }
3853 \@@_def_env:NNN p ( )
3854 \@@_def_env:NNN b [ ]
3855 \@@_def_env:NNN B \{ \}
3856 \@@_def_env:NNN v \vert \vert
3857 \@@_def_env:NNN V \Vert \Vert

```

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

### 13.1 Definition of `{pNiceMatrix}`

```

3858 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3859 {
3860   \bool_set_false:N \l_@@_preamble_bool
3861   \tl_clear:N \l_tmpa_tl
3862   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3863   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3864   \tl_put_right:Nn \l_tmpa_tl
3865   {
3866     *
3867     {
3868       \int_case:nnF \l_@@_last_col_int
3869       {
3870         { -2 } { \c@MaxMatrixCols }
3871         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3872       }
3873       { \int_eval:n { \l_@@_last_col_int - 1 } }
3874     }
3875     { #2 }
3876   }
3877   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }

```

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

```

3878 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3879 }
3880 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3881 \clist_map_inline:nn { p , b , B , v , V }
3882 {
3883   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3884   {
3885     \bool_gset_true:N \g_@@_delims_bool
3886     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3887     \int_if_zero:nT \l_@@_last_col_int
3888     {
3889       \bool_set_true:N \l_@@_last_col_without_value_bool
3890       \int_set:Nn \l_@@_last_col_int { -1 }
3891     }
3892     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3893     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3894   }
3895   { \use:c { end #1 NiceArray } }
3896 }

```

We define also an environment {NiceMatrix}

```

3897 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3898 {
3899   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3900   \int_if_zero:nT \l_@@_last_col_int
3901   {
3902     \bool_set_true:N \l_@@_last_col_without_value_bool
3903     \int_set:Nn \l_@@_last_col_int { -1 }
3904   }
3905   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3906   \bool_lazy_or:nnT
3907     \l_@@_except_borders_bool
3908     { \clist_if_empty_p:N \l_@@_vlines_clist }
3909     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3910   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3911 }
3912 { \endNiceArray }

```

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

```

3913 \cs_new_protected:Npn \@@_NotEmpty:
3914 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

## 13.2 The key renew-matrix

```

3915 \cs_set_protected:Npn \@@_renew_matrix:
3916 {
3917   \tl_map_inline:nn { pvVbB }
3918   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3919 }

```

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

```

3920 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3921 {

```

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.

```

3922   \dim_compare:nNt\l_@@_width_dim = \c_zero_dim
3923   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3924   \str_gset:Nn \g_@@_name_env_str { NiceTabular }

```

```

3925 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3926 \tl_if_empty:NF \l_@@_short_caption_tl
3927 {
3928   \tl_if_empty:NT \l_@@_caption_tl
3929   {
3930     \@@_error_or_warning:n { short-caption-without-caption }
3931     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3932   }
3933 }
3934 \tl_if_empty:NF \l_@@_label_tl
3935 {
3936   \tl_if_empty:NT \l_@@_caption_tl
3937   { \@@_error_or_warning:n { label-without-caption } }
3938 }
3939 \NewDocumentEnvironment { TabularNote } { b }
3940 {
3941   \bool_if:NTF \l_@@_in_code_after_bool
3942   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3943   {
3944     \tl_if_empty:NF \g_@@_tabularnote_tl
3945     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3946     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3947   }
3948 }
3949 { }
3950 \@@_settings_for_tabular:
3951 \NiceArray { #2 }
3952 }
3953 { \endNiceArray }
3954 \cs_new_protected:Npn \@@_settings_for_tabular:
3955 {
3956   \bool_set_true:N \l_@@_tabular_bool
3957   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3958   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3959   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3960 }

3961 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3962 {
3963   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3964   \dim_set:Nn \l_@@_width_dim { #1 }
3965   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3966   \@@_settings_for_tabular:
3967   \NiceArray { #3 }
3968 }
3969 {
3970   \endNiceArray
3971   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3972   { \@@_error:n { NiceTabularX~without~X } }
3973 }

3974 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3975 {
3976   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3977   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3978   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3979   \@@_settings_for_tabular:
3980   \NiceArray { #3 }
3981 }
3982 { \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`).

```

3983 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3984 {
3985   \bool_lazy_all:nT
3986   {
3987     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3988     { \l_@@_hvlines_bool }
3989     { ! \g_@@_delims_bool }
3990     { ! \l_@@_except_borders_bool }
3991   }
3992   {
3993     \bool_set_true:N \l_@@_except_borders_bool
3994     \clist_if_empty:NF \l_@@_corners_clist
3995     { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3996     \tl_gput_right:Nn \g_@@_rules_tl
3997     {
3998       \@@_stroke_block:nnnnn
3999       {
4000         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
4001         draw = \l_@@_rules_color_tl
4002       }
4003       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4004     }
4005   }
4006 }

4007 \cs_new_protected:Npn \@@_after_array:
4008 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

4009   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
4010   \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.

```

4011   \bool_if:NT \g_@@_last_col_found_bool
4012   { \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`.

```

4013   \bool_if:NT \l_@@_last_col_without_value_bool
4014   { \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.

```

4015   \bool_if:NT \l_@@_last_row_without_value_bool
4016   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4017 \tl_gput_right:Ne \g_@@_aux_tl
4018 {
4019   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4020   {
4021     \int_use:N \l_@@_first_row_int ,
4022     \int_use:N \c@iRow ,
4023     \int_use:N \g_@@_row_total_int ,
4024     \int_use:N \l_@@_first_col_int ,
4025     \int_use:N \c@jCol ,
4026     \int_use:N \g_@@_col_total_int
4027   }
4028 }
4029 \clist_if_empty:NF \g_@@_cbic_clist \@@_create_blocks_in_col:

```

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`).

```

4030 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4031 {
4032   \tl_gput_right:Ne \g_@@_aux_tl
4033   {
4034     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4035     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4036   }
4037 }
4038 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4039 {
4040   \tl_gput_right:Ne \g_@@_aux_tl
4041   {
4042     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4043     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4044     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4045     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4046   }
4047 }

```

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

```

4048 \@@_create_diag_nodes:

```

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

```

4049 \pgfpicture
4050 \@@_create_aliases_last:
4051 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4052 \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>12</sup>. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

4053 \bool_if:NT \l_@@_parallelize_diags_bool
4054 {
4055   \int_gzero:N \g_@@_ddots_int
4056   \int_gzero:N \g_@@_iddots_int

```

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

```

4057 \dim_gzero:N \g_@@_delta_x_one_dim
4058 \dim_gzero:N \g_@@_delta_y_one_dim
4059 \dim_gzero:N \g_@@_delta_x_two_dim

```

---

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

```

4060     \dim_gzero:N \g_@@_delta_y_two_dim
4061   }

4062   \bool_set_false:N \l_@@_initial_open_bool
4063   \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.

```

4064   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4065   \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

4066   \clist_if_empty:NF \l_@@_corners_clist
4067   {
4068     \bool_if:NTF \l_@@_no_cell_nodes_bool
4069     { \@@_error:n { corners~with~no~cell~nodes } }
4070     \@@_compute_corners:
4071   }

```

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

```

4072   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4073   \g_@@_pos_of_blocks_seq
4074   \g_@@_future_pos_of_blocks_seq
4075   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

4076   \@@_adjust_pos_of_blocks_seq:
4077   \@@_deal_with_rounded_corners:
4078   \legacy_if:nF { measuring@ } \@@_draw_rules:
4079   \tl_gclear:N \g_@@_rules_tl

```

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

```

4080   \IfPackageLoadedT { tikz }
4081   {
4082     \tikzset
4083     {
4084       every-picture / .style =
4085       {
4086         overlay ,
4087         remember~picture ,
4088         name~prefix = \@@_env: -
4089       }
4090     }
4091   }
4092   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4093   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4094   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4095   \cs_set_eq:NN \OverBrace \@@_OverBrace
4096   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4097   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4098   \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.

```
4099 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4100 \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.

```
4101 \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).

```
4102 \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`).

```
4103 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
4104 { \@@_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:`.

```
4105 \bool_set_true:N \l_@@_in_code_after_bool
4106 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4107 \scan_stop:
4108 \tl_gclear:N \g_nicematrix_code_after_tl
4109 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4110 \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.

```
4111 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4112 \tl_if_empty:NF \g_@@_pre_code_before_tl
4113 {
4114   \tl_gput_right:Ne \g_@@_aux_tl
4115   {
4116     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4117     { \exp_not:o \g_@@_pre_code_before_tl }
4118   }
4119   \tl_gclear:N \g_@@_pre_code_before_tl
4120 }
4121 \tl_if_empty:NF \g_nicematrix_code_before_tl
4122 {
4123   \tl_gput_right:Ne \g_@@_aux_tl
4124   {
4125     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4126     { \exp_not:o \g_nicematrix_code_before_tl }
4127   }
4128   \tl_gclear:N \g_nicematrix_code_before_tl
4129 }

4130 \str_gclear:N \g_@@_name_env_str
4131 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>13</sup>. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

---

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

```

4132 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4133 }

```

```

4134 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4135 {
4136   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4137   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

4138   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4139     { 0.6 \l_@@_xdots_shorten_start_dim }
4140   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4141     { 0.6 \l_@@_xdots_shorten_end_dim }
4142 }

```

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`.

```

4143 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4144 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4145 \cs_new_protected:Npn \@@_create_alias_nodes:
4146 {
4147   \int_step_inline:nn \c@iRow
4148   {
4149     \pgfnodealias
4150       { \l_@@_name_str - ##1 - last }
4151       { \@@_env: - ##1 - \int_use:N \c@jCol }
4152   }
4153   \int_step_inline:nn \c@jCol
4154   {
4155     \pgfnodealias
4156       { \l_@@_name_str - last - ##1 }
4157       { \@@_env: - \int_use:N \c@iRow - ##1 }
4158   }
4159   \pgfnodealias
4160     { \l_@@_name_str - last - last }
4161     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4162 }

```

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).

```

4163 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4164 {
4165   \seq_gset_map_e:Nnn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4166     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4167 }

```

The following command must *not* be protected.

```

4168 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4169 {
4170   { #1 }
4171   { #2 }
4172   {

```



```

4173     \int_compare:nNnTF { #3 } > { 98 }
4174     { \int_use:N \c@iRow }
4175     { #3 }
4176   }
4177   {
4178     \int_compare:nNnTF { #4 } > { 98 }
4179     { \int_use:N \c@jCol }
4180     { #4 }
4181   }
4182   { #5 }
4183 }

```

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).

```

4184 \AtBeginDocument
4185 {
4186   \cs_new_protected:Npe \@@_draw_dotted_lines:
4187   {
4188     \c_@@_pgfortikzpicture_tl
4189     \@@_draw_dotted_lines_i:
4190     \c_@@_endpgfortikzpicture_tl
4191   }
4192 }

```

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

```

4193 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4194 {
4195   \pgfrememberpicturepositiononpagetrue
4196   \pgf@relevantforpicturesizefalse
4197   \g_@@_HVdotsfor_lines_tl
4198   \g_@@_Vdots_lines_tl
4199   \g_@@_Ddots_lines_tl
4200   \g_@@_Iddots_lines_tl
4201   \g_@@_Cdots_lines_tl
4202   \g_@@_Ldots_lines_tl
4203 }

4204 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4205 {
4206   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4207   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4208 }

```

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

```

4209 \pgfdeclareshape { @@_diag_node }
4210 {
4211   \savedanchor { \five }
4212   {
4213     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4214     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4215   }
4216   \anchor { 5 } { \five }
4217   \anchor { center } { \pgfpointorigin }
4218   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4219   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4220   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4221   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4222   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }

```

```

4223 \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4224 \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4225 \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4226 \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4227 \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4228 }

```

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).

```

4229 \cs_new_protected:Npn \@@_create_diag_nodes:
4230 {
4231 \pgfpicture
4232 \pgfrememberpicturepositiononpagetrue
4233 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4234 {
4235 \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4236 \dim_set_eq:NN \l_tmpa_dim \pgf@x
4237 \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4238 \dim_set_eq:NN \l_tmpb_dim \pgf@y
4239 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4240 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4241 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4242 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4243 \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.

```

4244 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4245 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4246 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4247 \str_if_empty:NF \l_@@_name_str
4248 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4249 }

```

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

```

4250 \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4251 \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4252 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4253 \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4254 \pgfcoordinate
4255 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4256 \pgfnodealias
4257 { \@@_env: - last }
4258 { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4259 \str_if_empty:NF \l_@@_name_str
4260 {
4261 \pgfnodealias
4262 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4263 { \@@_env: - \int_use:N \l_tmpa_int }
4264 \pgfnodealias
4265 { \l_@@_name_str - last }
4266 { \@@_env: - last }
4267 }
4268 \endpgfpicture
4269 }

```

## 16 We draw the dotted lines

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

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

The command `\@@_find_extremities:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the  $x$ -value of the orientation vector of the line;
- the fourth argument is the  $y$ -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
{
  \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
  {
    \int_compare:nNnT { #4 } = { -1 }
    { \bool_set_true:N \l_@@_final_open_bool }
  }
  {
    \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
    {
```

```

        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
    }
}
}
\bool_if:NTF \l_@@_final_open_bool
{
    \int_sub:Nn \l_@@_final_i_int { #3 }
    \int_sub:Nn \l_@@_final_j_int { #4 }
    \bool_set_true:N \l_@@_stop_loop_bool
}
{
    \cs_if_exist:cTF
    {
        @@ _ dotted _
        \int_use:N \l_@@_final_i_int -
        \int_use:N \l_@@_final_j_int
    }
    {
        \int_sub:Nn \l_@@_final_i_int { #3 }
        \int_sub:Nn \l_@@_final_j_int { #4 }
        \bool_set_true:N \l_@@_final_open_bool
        \bool_set_true:N \l_@@_stop_loop_bool
    }
    {
        \cs_if_exist:cTF
        {
            pgf @ sh @ ns @ \@@_env:
            - \int_use:N \l_@@_final_i_int
            - \int_use:N \l_@@_final_j_int
        }
        { \bool_set_true:N \l_@@_stop_loop_bool }
        {
            \cs_set_nopar:cpn
            {
                @@ _ dotted _
                \int_use:N \l_@@_final_i_int -
                \int_use:N \l_@@_final_j_int
            }
            { }
        }
    }
}
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
    \int_sub:Nn \l_@@_initial_i_int { #3 }
    \int_sub:Nn \l_@@_initial_j_int { #4 }
    \bool_set_false:N \l_@@_initial_open_bool
    \int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
    {
        \int_compare:nNnTF { #3 } = { 1 }
        { \bool_set_true:N \l_@@_initial_open_bool }
        {
            \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
            { \bool_set_true:N \l_@@_initial_open_bool }
        }
    }
}
{
    \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
    {

```

```

\int_compare:nNtT { #4 } = { 1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
{
\int_compare:nNtT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
{
\int_compare:nNtT { #4 } = { -1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
}
}
\bool_if:NTF \l_@@_initial_open_bool
{
\int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool
}
{
\cs_if_exist:cTF
{
@@ _ dotted _
\int_use:N \l_@@_initial_i_int -
\int_use:N \l_@@_initial_j_int
}
{
\int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_initial_open_bool
\bool_set_true:N \l_@@_stop_loop_bool
}
{
\cs_if_exist:cTF
{
pgf @ sh @ ns @ \@@_env:
- \int_use:N \l_@@_initial_i_int
- \int_use:N \l_@@_initial_j_int
}
{ \bool_set_true:N \l_@@_stop_loop_bool }
{
\cs_set_nopar:cpn
{
@@ _ dotted _
\int_use:N \l_@@_initial_i_int -
\int_use:N \l_@@_initial_j_int
}
{ }
}
}
}
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
{ \int_use:N \l_@@_initial_i_int }
{ \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
{ \int_use:N \l_@@_final_i_int }
{ \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
{ }
}
}
}

```

The following version is slightly more efficient.

```

4270 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4271 {

```

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

```
4272 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4273 \l_@@_initial_i_int = #1
4274 \l_@@_initial_j_int = #2
4275 \l_@@_final_i_int = #1
4276 \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.

```
4277 \let \l_@@_stop_loop_bool \c_false_bool
4278 \bool_do_until:Nn \l_@@_stop_loop_bool
4279 {
```

We test if we are still in the matrix.

```
4280 \advance \l_@@_final_i_int by #3
4281 \advance \l_@@_final_j_int by #4
4282 \let \l_@@_final_open_bool \c_false_bool
4283 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4284 \if_int_compare:w #3 = \c_one_int
4285 \let \l_@@_final_open_bool \c_true_bool
4286 \else:
4287 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4288 \let \l_@@_final_open_bool \c_true_bool
4289 \fi:
4290 \fi:
4291 \else:
4292 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4293 \if_int_compare:w #4 = -1
4294 \let \l_@@_final_open_bool \c_true_bool
4295 \fi:
4296 \else:
4297 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4298 \if_int_compare:w #4 = \c_one_int
4299 \let \l_@@_final_open_bool \c_true_bool
4300 \fi:
4301 \fi:
4302 \fi:
4303 \fi:
4304 \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.

```
4305 {
```

We do a step backwards.

```
4306 \advance \l_@@_final_i_int by - #3
4307 \advance \l_@@_final_j_int by - #4
4308 \let \l_@@_stop_loop_bool \c_true_bool
4309 }
```

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`.

```
4310 {
4311 \cs_if_exist:cTF
4312 {
4313 @@ _ dotted _
4314 \int_use:N \l_@@_final_i_int -
4315 \int_use:N \l_@@_final_j_int
4316 }
4317 {
4318 \advance \l_@@_final_i_int by - #3
4319 \advance \l_@@_final_j_int by - #4
4320 \let \l_@@_final_open_bool \c_true_bool
```

```

4321         \let \l_@@_stop_loop_bool \c_true_bool
4322     }
4323     {
4324         \cs_if_exist:cTF
4325         {
4326             pgf @ sh @ ns @ \@@_env:
4327             - \int_use:N \l_@@_final_i_int
4328             - \int_use:N \l_@@_final_j_int
4329         }
4330         { \let \l_@@_stop_loop_bool \c_true_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4331     {
4332         \cs_set_nopar:cpn
4333         {
4334             @@ _ dotted _
4335             \int_use:N \l_@@_final_i_int -
4336             \int_use:N \l_@@_final_j_int
4337         }
4338         { }
4339     }
4340 }
4341 }
4342 }

```

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

```

4343     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4344     \l_tmpa_int = \l_@@_col_min_int
4345     \advance \l_tmpa_int by -1
4346     \bool_do_until:Nn \l_@@_stop_loop_bool
4347     {
4348         \advance \l_@@_initial_i_int by - #3
4349         \advance \l_@@_initial_j_int by - #4
4350         \let \l_@@_initial_open_bool \c_false_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4351     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4352     \if_int_compare:w #3 = \c_one_int
4353     \let \l_@@_initial_open_bool \c_true_bool
4354     \else:

```

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

```

4355         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4356         \let \l_@@_initial_open_bool \c_true_bool
4357         \fi:
4358     \fi:
4359     \else:
4360     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4361     \if_int_compare:w #4 = \c_one_int
4362     \let \l_@@_initial_open_bool \c_true_bool
4363     \fi:
4364     \else:
4365     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4366     \if_int_compare:w #4 = -1
4367     \let \l_@@_initial_open_bool \c_true_bool

```

```

4368         \fi:
4369     \fi:
4370 \fi:
4371 \fi:
4372 \bool_if:NTF \l_@@_initial_open_bool
4373 {
4374     \advance \l_@@_initial_i_int by #3
4375     \advance \l_@@_initial_j_int by #4
4376     \let \l_@@_stop_loop_bool \c_true_bool
4377 }
4378 {
4379     \cs_if_exist:cTF
4380     {
4381         @@ _ dotted _
4382         \int_use:N \l_@@_initial_i_int -
4383         \int_use:N \l_@@_initial_j_int
4384     }
4385     {
4386         \advance \l_@@_initial_i_int by #3
4387         \advance \l_@@_initial_j_int by #4
4388         \let \l_@@_initial_open_bool \c_true_bool
4389         \let \l_@@_stop_loop_bool \c_true_bool
4390     }
4391     {
4392         \cs_if_exist:cTF
4393         {
4394             pgf @ sh @ ns @ \@@_env:
4395             - \int_use:N \l_@@_initial_i_int
4396             - \int_use:N \l_@@_initial_j_int
4397         }
4398         { \let \l_@@_stop_loop_bool \c_true_bool }
4399         {
4400             \cs_set_nopar:cpn
4401             {
4402                 @@ _ dotted _
4403                 \int_use:N \l_@@_initial_i_int -
4404                 \int_use:N \l_@@_initial_j_int
4405             }
4406             { }
4407         }
4408     }
4409 }
4410 }

```

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.

```

4411 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4412 {
4413     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4414     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4415     { \int_use:N \l_@@_final_i_int }
4416     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4417     { }
4418 }
4419 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we known whether the extremities are closed or open) but before the analysis of



the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4420 \cs_new_protected:Npn \@@_open_shorten:
4421 {
4422   \bool_if:NT \l_@@_initial_open_bool
4423     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4424   \bool_if:NT \l_@@_final_open_bool
4425     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4426 }

```

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).

```

4427 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4428 {
4429   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4430   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4431   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4432   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

If there is no submatrices, we will speed up a little for the potential other dotted lines to draw.

```

4433   \seq_if_empty:NTF \g_@@_submatrix_seq
4434     { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4435     {

```

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`.

```

4436       \seq_map_inline:Nn \g_@@_submatrix_seq
4437       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4438     }
4439 }

```

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

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

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4440 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4441 {
4442   \if_int_compare:w #3 > #1
4443   \else:
4444     \if_int_compare:w #1 > #5
4445     \else:
4446       \if_int_compare:w #4 > #2
4447       \else:

```

```

4448         \if_int_compare:w #2 > #6
4449         \else:
4450             \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4451             \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4452             \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4453             \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4454         \fi:
4455     \fi:
4456 \fi:
4457 \fi:
4458 }

4459 \cs_new_protected:Npn \@@_set_initial_coords:
4460 {
4461     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4462     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4463 }
4464 \cs_new_protected:Npn \@@_set_final_coords:
4465 {
4466     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4467     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4468 }
4469 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4470 {
4471     \pgfpointanchor
4472     {
4473         \@@_env:
4474         - \int_use:N \l_@@_initial_i_int
4475         - \int_use:N \l_@@_initial_j_int
4476     }
4477     { #1 }
4478     \@@_set_initial_coords:
4479 }
4480 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4481 {
4482     \pgfpointanchor
4483     {
4484         \@@_env:
4485         - \int_use:N \l_@@_final_i_int
4486         - \int_use:N \l_@@_final_j_int
4487     }
4488     { #1 }
4489     \@@_set_final_coords:
4490 }

4491 \cs_new_protected:Npn \@@_open_x_initial_dim:
4492 {
4493     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4494     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4495     {
4496         \cs_if_exist:cT
4497         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4498         {
4499             \pgfpointanchor
4500             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4501             { west }
4502             \dim_set:Nn \l_@@_x_initial_dim
4503             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4504         }
4505     }

```

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

```

4506     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4507     {

```

```

4508     \l_@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4509     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4510     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4511   }
4512 }
4513 \cs_new_protected:Npn \l_@@_open_x_final_dim:
4514 {
4515   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4516   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4517   {
4518     \cs_if_exist:cT
4519     { \pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4520     {
4521       \pgfpointanchor
4522       { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4523       { east }
4524       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4525       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4526     }
4527   }

```

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

```

4528   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4529   {
4530     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4531     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4532     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4533   }
4534 }

```

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.

```

4535 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4536 {
4537   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4538   \cs_if_free:cT { \l_@@_dotted_ #1 - #2 }
4539   {
4540     \l_@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4541   \bool_if:NT \g_@@_aux_found_bool
4542   {
4543     {
4544       \l_@@_open_shorten:
4545       \int_if_zero:nTF { #1 }
4546       { \color { nicematrix-first-row } }
4547     }

```

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.

```

4548     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4549     { \color { nicematrix-last-row } }
4550   }
4551   \keys_set:nn { nicematrix / xdots } { #3 }
4552   \l_@@_color:o \l_@@_xdots_color_tl
4553   \l_@@_actually_draw_Ldots:
4554 }
4555 }
4556 }
4557 }

```

The command `\l_@@_actually_draw_Ldots:` has the following implicit arguments:

- \l\_@@\_initial\_i\_int
- \l\_@@\_initial\_j\_int
- \l\_@@\_initial\_open\_bool
- \l\_@@\_final\_i\_int
- \l\_@@\_final\_j\_int
- \l\_@@\_final\_open\_bool.

The following function is also used by \Hdotsfor.

```

4558 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4559 {
4560   \bool_if:NTF \l_@@_initial_open_bool
4561   {
4562     \@@_open_x_initial_dim:
4563     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4564     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4565   }
4566   { \@@_set_initial_coords_from_anchor:n { base-east } }
4567   \bool_if:NTF \l_@@_final_open_bool
4568   {
4569     \@@_open_x_final_dim:
4570     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4571     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4572   }
4573   { \@@_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.

```

4574   \bool_lazy_all:nTF
4575   {
4576     \l_@@_initial_open_bool
4577     \l_@@_final_open_bool
4578     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4579   }
4580   {
4581     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4582     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4583   }

```

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 (?).

```

4584   {
4585     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4586     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4587   }
4588   \@@_draw_line:
4589 }

```

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.

```

4590 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4591 {
4592   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4593   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4594   {
4595     \@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4596     \bool_if:NT \g_@@_aux_found_bool
4597     {
4598     {
4599         \@@_open_shorten:
4600         \int_if_zero:nTF { #1 }
4601         { \color { nicematrix-first-row } }
4602         {

```

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.

```

4603         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4604         { \color { nicematrix-last-row } }
4605     }
4606     \keys_set:nn { nicematrix / xdots } { #3 }
4607     \@@_color:o \l_@@_xdots_color_tl
4608     \@@_actually_draw_Cdots:
4609 }
4610 }
4611 }
4612 }

```

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`.

```

4613 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4614 {
4615     \bool_if:NTF \l_@@_initial_open_bool
4616     \@@_open_x_initial_dim:
4617     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4618     \bool_if:NTF \l_@@_final_open_bool
4619     \@@_open_x_final_dim:
4620     { \@@_set_final_coords_from_anchor:n { mid-west } }
4621     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4622     {
4623         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4624         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4625         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4626         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4627         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4628     }
4629     {
4630         \bool_if:NT \l_@@_initial_open_bool
4631         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4632         \bool_if:NT \l_@@_final_open_bool
4633         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4634     }
4635     \@@_draw_line:
4636 }
4637 \cs_new_protected:Npn \@@_open_y_initial_dim:
4638 {
4639     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4640     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int

```

```

4641 {
4642   \cs_if_exist:cT
4643     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4644     {
4645       \pgfpointanchor
4646         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4647         { north }
4648       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4649       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4650     }
4651   }
4652   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4653   {
4654     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4655     \dim_set:Nn \l_@@_y_initial_dim
4656       {
4657         \fp_to_dim:n
4658         {
4659           \pgf@y
4660           + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4661         }
4662       }
4663   }
4664 }
4665 \cs_new_protected:Npn \@@_open_y_final_dim:
4666 {
4667   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4668   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4669   {
4670     \cs_if_exist:cT
4671       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4672       {
4673         \pgfpointanchor
4674           { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4675           { south }
4676         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4677         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4678       }
4679   }
4680   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4681   {
4682     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4683     \dim_set:Nn \l_@@_y_final_dim
4684       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4685   }
4686 }

```

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.

```

4687 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4688 {
4689   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4690   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4691   {
4692     \@@_find_extremities:nnnn { #1 } { #2 } 1 0

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4693   \bool_if:NT \g_@@_aux_found_bool
4694   {
4695     {
4696       \@@_open_shorten:
4697       \int_if_zero:nTF { #2 }
4698       { \color { nicematrix-first-col } }

```

```

4699         {
4700             \int_compare:nNt { #2 } = \l_@@_last_col_int
4701             { \color { nicematrix-last-col } }
4702         }
4703         \keys_set:nn { nicematrix / xdots } { #3 }
4704         \@@_color:o \l_@@_xdots_color_tl
4705         \bool_if:NTF \l_@@_Vbrace_bool
4706             \@@_actually_draw_Vbrace:
4707             \@@_actually_draw_Vdots:
4708     }
4709 }
4710 }
4711 }

```

The following function is used by regular calls of `\Vdots` or `\Vdotsfor` but not by `\Vbrace`. The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4712 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4713 {
4714     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4715     \@@_actually_draw_Vdots_i:
4716     \@@_actually_draw_Vdots_ii:
4717     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4718     \@@_draw_line:
4719 }

```

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

```

4720 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4721 {
4722     \@@_open_y_initial_dim:
4723     \@@_open_y_final_dim:
4724     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4725     {
4726         \@@_qpoint:n { col - 1 }
4727         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4728         \dim_sub:Nn \l_@@_x_initial_dim
4729         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4730     }
4731     {
4732         \bool_lazy_and:nnTF
4733         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4734         { \int_compare_p:nNn \l_@@_initial_j_int = \g_@@_col_total_int }

```

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

```

4735     {
4736         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4737         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4738         \dim_add:Nn \l_@@_x_initial_dim
4739         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4740     }

```

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

```

4741     {
4742         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4743         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4744         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4745         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4746     }
4747 }
4748 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the  $x$ -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

4749 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4750 {
4751     \bool_set_false:N \l_tmpa_bool
4752     \bool_if:NF \l_@@_initial_open_bool
4753     {
4754         \bool_if:NF \l_@@_final_open_bool
4755         {
4756             \@@_set_initial_coords_from_anchor:n { south-west }
4757             \@@_set_final_coords_from_anchor:n { north-west }
4758             \bool_set:Nn \l_tmpa_bool
4759                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4760         }
4761     }

```

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

```

4762     \bool_if:NTF \l_@@_initial_open_bool
4763     {
4764         \@@_open_y_initial_dim:
4765         \@@_set_final_coords_from_anchor:n { north }
4766         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4767     }
4768     {
4769         \@@_set_initial_coords_from_anchor:n { south }
4770         \bool_if:NTF \l_@@_final_open_bool
4771             \@@_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.

```

4772     {
4773         \@@_set_final_coords_from_anchor:n { north }
4774         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4775         {
4776             \dim_set:Nn \l_@@_x_initial_dim
4777             {
4778                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4779                     \l_@@_x_initial_dim \l_@@_x_final_dim
4780             }
4781         }
4782     }
4783 }
4784 }

```

The following function is used by `\Vbrace` but not by regular uses of `\Vdots` or `\Vdotsfor`.

The command `\@@_actually_draw_Vbrace:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`



- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4785 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4786 {
4787   \bool_if:NTF \l_@@_initial_open_bool
4788     \@@_open_y_initial_dim:
4789     { \@@_set_initial_coords_from_anchor:n { south } }
4790   \bool_if:NTF \l_@@_final_open_bool
4791     \@@_open_y_final_dim:
4792     { \@@_set_final_coords_from_anchor:n { north } }

```

Now, we have the correct values for the  $y$ -values of both extremities of the brace. We have to compute the  $x$ -value (there is only one  $x$ -value since, of course, the brace is vertical).

If we are in the first (exterior) column, the brace must be drawn right flush.

```

4793   \int_if_zero:nTF \l_@@_initial_j_int
4794   {
4795     \@@_qpoint:n { col - 1 }
4796     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4797     \dim_sub:Nn \l_@@_x_initial_dim
4798     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4799   }

```

Elsewhere, the brace must be drawn left flush.

```

4800   {
4801     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4802     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4803     \dim_add:Nn \l_@@_x_initial_dim
4804     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4805   }

```

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

```

4806   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4807   \@@_draw_line:
4808 }

```

```

4809 \cs_new:Npn \@@_colsep:
4810 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4811 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4812 {
4813   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4814   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4815   {
4816     \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4817   \bool_if:NT \g_@@_aux_found_bool
4818   {
4819     {
4820       \@@_open_shorten:
4821       \keys_set:nn { nicematrix / xdots } { #3 }

```

```

4822         \l_@@_xdots_color_tl
4823         \l_@@_actually_draw_Ddots:
4824     }
4825 }
4826 }
4827 }

```

The command `\l_@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

```

4828 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4829 {
4830     \bool_if:NTF \l_@@_initial_open_bool
4831     {
4832         \l_@@_open_y_initial_dim:
4833         \l_@@_open_x_initial_dim:
4834     }
4835     { \l_@@_set_initial_coords_from_anchor:n { south-east } }
4836     \bool_if:NTF \l_@@_final_open_bool
4837     {
4838         \l_@@_open_x_final_dim:
4839         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4840     }
4841     { \l_@@_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.

```

4842     \bool_if:NT \l_@@_parallelize_diags_bool
4843     {
4844         \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).

```

4845         \int_compare:nNnTF \g_@@_ddots_int = 1

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4846     {
4847         \dim_gset:Nn \g_@@_delta_x_one_dim
4848         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4849         \dim_gset:Nn \g_@@_delta_y_one_dim
4850         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4851     }

```

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`.

```

4852     {
4853         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4854         {
4855             \dim_set:Nn \l_@@_y_final_dim
4856             {
4857                 \l_@@_y_initial_dim +
4858                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4859                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4860             }

```

```

4861         }
4862     }
4863 }
4864 \@@_draw_line:
4865 }

```

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.

```

4866 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4867 {
4868     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4869     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4870     {
4871         \@@_find_extremities:nnnn { #1 } { #2 } { 1 } { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4872         \bool_if:NT \g_@@_aux_found_bool
4873         {
4874             {
4875                 \@@_open_shorten:
4876                 \keys_set:nn { nicematrix / xdots } { #3 }
4877                 \@@_color:o \l_@@_xdots_color_tl
4878                 \@@_actually_draw_Iddots:
4879             }
4880         }
4881     }
4882 }

```

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`.

```

4883 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4884 {
4885     \bool_if:NTF \l_@@_initial_open_bool
4886     {
4887         \@@_open_y_initial_dim:
4888         \@@_open_x_initial_dim:
4889     }
4890     { \@@_set_initial_coords_from_anchor:n { south~west } }
4891     \bool_if:NTF \l_@@_final_open_bool
4892     {
4893         \@@_open_y_final_dim:
4894         \@@_open_x_final_dim:
4895     }
4896     { \@@_set_final_coords_from_anchor:n { north~east } }
4897     \bool_if:NT \l_@@_parallelize_diags_bool
4898     {
4899         \int_gincr:N \g_@@_iddots_int
4900         \int_compare:nNnTF \g_@@_iddots_int = 1
4901         {
4902             \dim_gset:Nn \g_@@_delta_x_two_dim

```

```

4903         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4904     \dim_gset:Nn \g_@@_delta_y_two_dim
4905         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4906     }
4907     {
4908         \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4909         {
4910             \dim_set:Nn \l_@@_y_final_dim
4911             {
4912                 \l_@@_y_initial_dim +
4913                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4914                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4915             }
4916         }
4917     }
4918 }
4919 \@@_draw_line:
4920 }

```

## 17 The actual instructions for drawing the dotted lines with TikZ

The command `\@@_draw_line:` 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`

```

4921 \cs_new_protected:Npn \@@_draw_line:
4922 {
4923     \pgfrememberpicturepositiononpagetrue
4924     \pgf@relevantforpicturesizefalse
4925     \bool_lazy_or:nnTF
4926         \l_@@_dotted_bool
4927         { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4928         \@@_draw_standard_dotted_line:
4929         \@@_draw_unstandard_dotted_line:
4930 }

```

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.

```

4931 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4932 {
4933     \begin { scope }
4934     \@@_draw_unstandard_dotted_line:o
4935         { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4936     }

```

We have used the fact that, in PGF, a 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.

```

4937 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4938 {
4939   \@@_draw_unstandard_dotted_line:nooo
4940   { #1 }
4941   \l_@@_xdots_up_tl
4942   \l_@@_xdots_down_tl
4943   \l_@@_xdots_middle_tl
4944 }
4945 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following TikZ styles are for the three labels (set by the symbols  $_$ ,  $\wedge$  and  $=$ ) of a continuous line with a non-standard style.

```

4946 \AtBeginDocument
4947 {
4948   \IfPackageLoadedT { tikz }
4949   {
4950     \tikzset
4951     {
4952       @@_node_above / .style = { sloped , above } ,
4953       @@_node_below / .style = { sloped , below } ,
4954       @@_node_middle / .style =
4955       {
4956         sloped ,
4957         inner~sep = \c_@@_innersep_middle_dim
4958       }
4959     }
4960   }
4961 }

4962 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4963 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4964 \dim_zero_new:N \l_@@_l_dim
4965 \dim_set:Nn \l_@@_l_dim
4966 {
4967   \fp_to_dim:n
4968   {
4969     sqrt
4970     (
4971       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4972       +
4973       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4974     )
4975   }
4976 }

```

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.

```

4977 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4978 {
4979   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4980   \@@_draw_unstandard_dotted_line_i:
4981 }

```

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

```

4982 \bool_if:NT \l_@@_xdots_h_labels_bool
4983 {
4984   \tikzset
4985   {
4986     @@_node_above / .style = { auto = left } ,
4987     @@_node_below / .style = { auto = right } ,
4988     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4989   }
4990 }
4991 \tl_if_empty:nF { #4 }
4992 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4993 \dim_zero:N \l_tmpa_dim
4994 \dim_zero:N \l_tmpb_dim
4995 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4996 {

```

We test whether the brace is vertical or horizontal.

```

4997   \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4998   { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4999   { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5000 }
5001 {
5002   \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5003   {
5004     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5005     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5006     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5007   }
5008 }
5009 \use:e
5010 {
5011   \exp_not:N \begin { scope }
5012   [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5013   }
5014 \draw
5015 [ #1 ]
5016 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5017 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
5018   node [ @@_node_below ] { $ \scriptstyle #3 $ }
5019   node [ @@_node_above ] { $ \scriptstyle #2 $ }
5020 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5021 \end { scope }
5022 \end { scope }
5023 }
5024 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5025 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5026 {
5027   \dim_set:Nn \l_tmpa_dim
5028   {
5029     \l_@@_x_initial_dim
5030     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5031     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5032   }
5033   \dim_set:Nn \l_tmpb_dim
5034   {
5035     \l_@@_y_initial_dim
5036     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5037     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5038   }
5039   \dim_set:Nn \l_@@_tmpc_dim
5040   {
5041     \l_@@_x_final_dim

```

```

5042     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5043     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5044   }
5045   \dim_set:Nn \l_@@_tmpd_dim
5046   {
5047     \l_@@_y_final_dim
5048     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5049     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5050   }
5051   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5052   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5053   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5054   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5055 }

```

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

```

5056 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5057 {
5058 {

```

The dimension `\l_@@_l_dim` is the length  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

5059   \dim_zero_new:N \l_@@_l_dim
5060   \dim_set:Nn \l_@@_l_dim
5061   {
5062     \fp_to_dim:n
5063     {
5064       sqrt
5065       (
5066         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5067         +
5068         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5069       )
5070     }
5071   }

```

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.

```

5072   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5073   {
5074     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5075     \@@_draw_standard_dotted_line_i:
5076   }
5077 }
5078 \bool_lazy_all:nF
5079 {
5080   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5081   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5082   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5083 }
5084 \@@_labels_standard_dotted_line:
5085 }
5086 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5087 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5088 {

```

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

```

5089   \int_set:Nn \l_tmpa_int
5090   {

```

```

5091 \dim_ratio:nn
5092 {
5093   \l_@@_l_dim
5094   - \l_@@_xdots_shorten_start_dim
5095   - \l_@@_xdots_shorten_end_dim
5096 }
5097 \l_@@_xdots_inter_dim
5098 }

```

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

```

5099 \dim_set:Nn \l_tmpa_dim
5100 {
5101   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5102   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5103 }
5104 \dim_set:Nn \l_tmpb_dim
5105 {
5106   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5107   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5108 }

```

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.

```

5109 \dim_gadd:Nn \l_@@_x_initial_dim
5110 {
5111   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5112   \dim_ratio:nn
5113   {
5114     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5115     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5116   }
5117   { 2 \l_@@_l_dim }
5118 }
5119 \dim_gadd:Nn \l_@@_y_initial_dim
5120 {
5121   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5122   \dim_ratio:nn
5123   {
5124     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5125     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5126   }
5127   { 2 \l_@@_l_dim }
5128 }
5129 \pgf@relevantforpicturesizefalse
5130 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5131 {
5132   \pgfpathcircle
5133   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5134   \l_@@_xdots_radius_dim
5135   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5136   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5137 }
5138 \pgfusepathqfill
5139 }

```

```

5140 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5141 {
5142   \pgfscope
5143   \pgftransformshift
5144   {
5145     \pgfpointlineatime { 0.5 }
5146     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }

```



```

5147         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5148     }
5149     \fp_set:Nn \l_tmpa_fp
5150     {
5151         atand
5152         (
5153             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5154             \l_@@_x_final_dim - \l_@@_x_initial_dim
5155         )
5156     }
5157     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5158     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5159     \tl_if_empty:NF \l_@@_xdots_middle_tl
5160     {
5161         \begin { pgfscope }
5162         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5163         \pgfnode
5164         { rectangle }
5165         { center }
5166         {
5167             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5168             {
5169                 $ % $
5170                 \scriptstyle \l_@@_xdots_middle_tl
5171                 $ % $
5172             }
5173         }
5174         { }
5175         {
5176             \pgfsetfillcolor { white }
5177             \pgfusepath { fill }
5178         }
5179         \end { pgfscope }
5180     }
5181     \tl_if_empty:NF \l_@@_xdots_up_tl
5182     {
5183         \pgfnode
5184         { rectangle }
5185         { south }
5186         {
5187             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5188             {
5189                 $ % $
5190                 \scriptstyle \l_@@_xdots_up_tl
5191                 $ % $
5192             }
5193         }
5194         { }
5195         { \pgfusepath { } }
5196     }
5197     \tl_if_empty:NF \l_@@_xdots_down_tl
5198     {
5199         \pgfnode
5200         { rectangle }
5201         { north }
5202         {
5203             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5204             {
5205                 $ % $
5206                 \scriptstyle \l_@@_xdots_down_tl
5207                 $ % $
5208             }
5209         }
5210     }

```

```

5210         { }
5211         { \pgfusepath { } }
5212     }
5213 \endpgfscope
5214 }

```

## 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 `\AtBeginDocument` and the *arg spec* will be rescanned.

```

5215 \AtBeginDocument
5216 {

```

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

```

5217 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5218 \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5219 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5220 {
5221     \int_if_zero:nTF \c@jCol
5222     { \@@_error:nn { in~first~col } { \Ldots } }
5223     {
5224         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5225         { \@@_error:nn { in~last~col } { \Ldots } }
5226         {
5227             \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5228             { #1 , down = #2 , up = #3 , middle = #4 }
5229         }
5230     }
5231     \bool_if:NF \l_@@_nullify_dots_bool
5232     { \phantom { \ensuremath { \@@_old_ldots: } } }
5233     \bool_gset_true:N \g_@@_empty_cell_bool
5234 }

```

```

5235 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5236 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5237 {
5238     \int_if_zero:nTF \c@jCol
5239     { \@@_error:nn { in~first~col } { \Cdots } }
5240     {
5241         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5242         { \@@_error:nn { in~last~col } { \Cdots } }
5243         {
5244             \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5245             { #1 , down = #2 , up = #3 , middle = #4 }
5246         }
5247     }
5248     \bool_if:NF \l_@@_nullify_dots_bool
5249     { \phantom { \ensuremath { \@@_old_cdots: } } }
5250     \bool_gset_true:N \g_@@_empty_cell_bool
5251 }

```

```

5252 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5253 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5254 {
5255   \int_if_zero:nTF \c@iRow
5256   { \@@_error:nn { in~first~row } { \Vdots } }
5257   {
5258     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5259     { \@@_error:nn { in~last~row } { \Vdots } }
5260     {
5261       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5262       { #1 , down = #2 , up = #3 , middle = #4 }
5263     }
5264   }
5265   \bool_if:NF \l_@@_nullify_dots_bool
5266   { \phantom { \ensuremath { \@@_old_vdots: } } }
5267   \bool_gset_true:N \g_@@_empty_cell_bool
5268 }

5269 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5270 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5271 {
5272   \int_case:nnF \c@iRow
5273   {
5274     0 { \@@_error:nn { in~first~row } { \Ddots } }
5275     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5276   }
5277   {
5278     \int_case:nnF \c@jCol
5279     {
5280       0 { \@@_error:nn { in~first~col } { \Ddots } }
5281       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5282     }
5283     {
5284       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5285       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5286       { #1 , down = #2 , up = #3 , middle = #4 }
5287     }
5288   }
5289 }
5290 \bool_if:NF \l_@@_nullify_dots_bool
5291 { \phantom { \ensuremath { \@@_old_ddots: } } }
5292 \bool_gset_true:N \g_@@_empty_cell_bool
5293 }

5294 \cs_new_protected:Npn \@@_Iddots:
5295 { \@@_collect_options:n { \@@_Iddots_i } }
5296 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5297 {
5298   \int_case:nnF \c@iRow
5299   {
5300     0 { \@@_error:nn { in~first~row } { \Iddots } }
5301     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5302   }
5303   {
5304     \int_case:nnF \c@jCol
5305     {
5306       0 { \@@_error:nn { in~first~col } { \Iddots } }
5307       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5308     }
5309     {
5310       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5311       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }

```

```

5312         { #1 , down = #2 , up = #3 , middle = #4 }
5313     }
5314 }
5315 \bool_if:NF \l_@@_nullify_dots_bool
5316 { \phantom { \ensuremath { \@@_old_iddots: } } }
5317 \bool_gset_true:N \g_@@_empty_cell_bool
5318 }
5319 }

```

End of the `\AddToHook`.

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

```

5320 \keys_define:nn { nicematrix / Ddots }
5321 {
5322     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5323     draw-first .value_forbidden:n = true ,
5324 }

```

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

```

5325 \cs_new_protected:Npn \@@_Hspace:
5326 {
5327     \bool_gset_true:N \g_@@_empty_cell_bool
5328     \hspace
5329 }

```

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`.

```

5330 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. TikZ nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5331 \cs_new:Npn \@@_Hdotsfor:
5332 {
5333     \bool_lazy_and:nnTF
5334     { \int_if_zero_p:n \c@jCol }
5335     { \int_if_zero_p:n \l_@@_first_col_int }
5336     {
5337         \bool_if:NTF \g_@@_after_col_zero_bool
5338         {
5339             \multicolumn { 1 } { c } { }
5340             \@@_Hdotsfor_i:
5341         }
5342         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5343     }
5344     {
5345         \multicolumn { 1 } { c } { }
5346         \@@_Hdotsfor_i:
5347     }
5348 }

```

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:`).

```

5349 \AtBeginDocument
5350 {

```

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

```

5351     \cs_new_protected:Npn \@@_Hdotsfor_i:
5352     { \@@_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`).

```

5353 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5354 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5355 {
5356   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5357   {
5358     \@@_Hdotsfor:nnnn
5359     { \int_use:N \c@iRow }
5360     { \int_use:N \c@jCol }
5361     { #2 }
5362     {
5363       #1 , #3 ,
5364       down = \exp_not:n { #4 } ,
5365       up = \exp_not:n { #5 } ,
5366       middle = \exp_not:n { #6 }
5367     }
5368   }
5369   \prg_replicate:nn { #2 - 1 }
5370   {
5371     &
5372     \multicolumn { 1 } { c } { }
5373     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5374   }
5375 }
5376 }

```

```

5377 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5378 {
5379   \bool_set_false:N \l_@@_initial_open_bool
5380   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5381 \int_set:Nn \l_@@_initial_i_int { #1 }
5382 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5383 \int_compare:nNnTF { #2 } = 1
5384 {
5385   \int_set:Nn \l_@@_initial_j_int 1
5386   \bool_set_true:N \l_@@_initial_open_bool
5387 }
5388 {
5389   \cs_if_exist:cTF
5390   {
5391     pgf @ sh @ ns @ \@@_env:
5392     - \int_use:N \l_@@_initial_i_int
5393     - \int_eval:n { #2 - 1 }
5394   }
5395   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5396   {
5397     \int_set:Nn \l_@@_initial_j_int { #2 }
5398     \bool_set_true:N \l_@@_initial_open_bool
5399   }
5400 }
5401 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5402 {
5403   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5404   \bool_set_true:N \l_@@_final_open_bool
5405 }
5406 {
5407   \cs_if_exist:cTF
5408   {
5409     pgf @ sh @ ns @ \@@_env:

```

```

5410         - \int_use:N \l_@@_final_i_int
5411         - \int_eval:n { #2 + #3 }
5412     }
5413     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5414     {
5415         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5416         \bool_set_true:N \l_@@_final_open_bool
5417     }
5418 }
5419 \bool_if:NT \g_@@_aux_found_bool
5420 {
5421     {
5422         \@@_open_shorten:
5423         \int_if_zero:nTF { #1 }
5424         { \color { nicematrix-first-row } }
5425         {
5426             \int_compare:nNnT { #1 } = \g_@@_row_total_int
5427             { \color { nicematrix-last-row } }
5428         }
5429         \keys_set:nn { nicematrix / xdots } { #4 }
5430         \@@_color:o \l_@@_xdots_color_tl
5431         \@@_actually_draw_Ldots:
5432     }
5433 }

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5434 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5435 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5436 }

```

```

5437 \AtBeginDocument
5438 {
5439     \cs_new_protected:Npn \@@_Vdotsfor:
5440     { \@@_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`).

```

5441 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5442 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5443 {
5444     \bool_gset_true:N \g_@@_empty_cell_bool
5445     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5446     {
5447         \@@_Vdotsfor:nnnn
5448         { \int_use:N \c@iRow }
5449         { \int_use:N \c@jCol }
5450         { #2 }
5451         {
5452             #1 , #3 ,
5453             down = \exp_not:n { #4 } ,
5454             up = \exp_not:n { #5 } ,
5455             middle = \exp_not:n { #6 }
5456         }
5457     }
5458 }
5459 }

```

**#1** is the number of row;  
**#2** is the number of column;

#3 is the numbers of rows which are involved;

```

5460 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5461 {
5462   \bool_set_false:N \l_@@_initial_open_bool
5463   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5464   \int_set:Nn \l_@@_initial_j_int { #2 }
5465   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5466   \int_compare:nNnTF { #1 } = 1
5467   {
5468     \int_set:Nn \l_@@_initial_i_int 1
5469     \bool_set_true:N \l_@@_initial_open_bool
5470   }
5471   {
5472     \cs_if_exist:cTF
5473     {
5474       pgf @ sh @ ns @ \@@_env:
5475       - \int_eval:n { #1 - 1 }
5476       - \int_use:N \l_@@_initial_j_int
5477     }
5478     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5479     {
5480       \int_set:Nn \l_@@_initial_i_int { #1 }
5481       \bool_set_true:N \l_@@_initial_open_bool
5482     }
5483   }
5484   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5485   {
5486     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5487     \bool_set_true:N \l_@@_final_open_bool
5488   }
5489   {
5490     \cs_if_exist:cTF
5491     {
5492       pgf @ sh @ ns @ \@@_env:
5493       - \int_eval:n { #1 + #3 }
5494       - \int_use:N \l_@@_final_j_int
5495     }
5496     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5497     {
5498       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5499       \bool_set_true:N \l_@@_final_open_bool
5500     }
5501   }
5502   \bool_if:NT \g_@@_aux_found_bool
5503   {
5504     {
5505       \@@_open_shorten:
5506       \int_if_zero:nTF { #2 }
5507       { \color { nicematrix-first-col } }
5508       {
5509         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5510         { \color { nicematrix-last-col } }
5511       }
5512       \keys_set:nn { nicematrix / xdots } { #4 }
5513       \@@_color:o \l_@@_xdots_color_tl
5514       \bool_if:NTF \l_@@_Vbrace_bool
5515       \@@_actually_draw_Vbrace:
5516       \@@_actually_draw_Vdots:
5517     }
5518   }

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5519 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5520 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5521 }

```

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

```

5522 \NewDocumentCommand \@@_rotate: { 0 { } }
5523 {
5524   \bool_gset_true:N \g_@@_rotate_bool
5525   \keys_set:nn { nicematrix / rotate } { #1 }
5526   \ignorespaces
5527 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5528 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5529 \keys_define:nn { nicematrix / rotate }
5530 {
5531   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5532   c .value_forbidden:n = true ,
5533   -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5534   -90 .value_forbidden:n = true ,
5535   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5536 }

```

## 19 The command `\line` accessible in `\CodeAfter`

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).<sup>14</sup>

```

5537 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5538 {
5539   \tl_if_empty:nTF { #2 }
5540   { #1 }
5541   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5542 }
5543 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5544 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

---

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



With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```
5545 \AtBeginDocument
5546 {
```

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

```
5547 \tl_set_rescan:Nnn \l_tmpa_tl { }
5548 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5549 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5550 {
5551 {
5552 \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5553 \@@_color:o \l_@@_xdots_color_tl
5554 \use:e
5555 {
5556 \@@_line_i:nn
5557 { \@@_double_int_eval:n #2 - \q_stop }
5558 { \@@_double_int_eval:n #3 - \q_stop }
5559 }
5560 }
5561 }
5562 }

5563 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5564 {
5565 \bool_set_false:N \l_@@_initial_open_bool
5566 \bool_set_false:N \l_@@_final_open_bool
5567 \bool_lazy_or:nnTF
5568 { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5569 { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5570 { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }
```

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

```
5571 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5572 }

5573 \AtBeginDocument
5574 {
5575 \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5576 {
```

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:.`

```
5577 \c_@@_pgfortikzpicture_tl
5578 \@@_draw_line_iii:nn { #1 } { #2 }
5579 \c_@@_endpgfortikzpicture_tl
5580 }
5581 }
```

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

```
5582 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5583 {
5584 \pgfrememberpicturepositiononpagetrue
5585 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5586 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5587 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5588 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5589 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5590 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5591 \@@_draw_line:
5592 }
```

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.

```
5593 \cs_new:Npn @@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5594 \cs_new:Npn @@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }
```

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

```
5595 \cs_set_protected:Npn @@_put_in_row_style:n #1
5596 {
5597   \tl_gput_right:Ne \g_@@_row_style_tl
5598   {
```

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

```
5599   \exp_not:N
5600   @@_if_row_less_than:nn
5601   { \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`).

```
5602   {
5603     \exp_not:N
5604     @@_if_col_greater_than:nn
5605     { \int_use:N \c@jCol }
5606     { \exp_not:n { #1 } \scan_stop: }
5607   }
5608 }
5609 }
5610 \cs_generate_variant:Nn @@_put_in_row_style:n { e }
```

```
5611 \keys_define:nn { nicematrix / RowStyle }
5612 {
5613   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5614   cell-space-top-limit+ .code:n =
5615     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5616   cell-space-top-limit+ .value_required:n = true ,
5617   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5618   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5619   cell-space-bottom-limit+ .code:n =
5620     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5621   cell-space-bottom-limit+ .value_required:n = true ,
5622   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5623   cell-space-limits .meta:n =
5624     {
5625       cell-space-top-limit = #1 ,
```

```

5626     cell-space-bottom-limit = #1 ,
5627   } ,
5628   cell-space-limits+ .meta:n =
5629   {
5630     cell-space-top-limit += #1 ,
5631     cell-space-bottom-limit += #1 ,
5632   } ,
5633   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5634   color .tl_set:N = \l_@@_color_tl ,
5635   color .value_required:n = true ,
5636   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5637   nb-rows .code:n =
5638     \str_if_eq:eeTF { #1 } { * }
5639     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5640     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5641   nb-rows .value_required:n = true ,
5642   fill .tl_set:N = \l_@@_fill_tl ,
5643   fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

5644   opacity .tl_set:N = \l_@@_opacity_tl ,
5645   opacity .value_required:n = true ,
5646   rowcolor .tl_set:N = \l_@@_fill_tl ,
5647   rowcolor .value_required:n = true ,
5648   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5649   rounded-corners .default:n = 4 pt ,
5650   unknown .code:n =
5651     \@@_unknown_key:nn
5652     { nicematrix / RowStyle }
5653     { Unknown-key-for-RowStyle }
5654 }

```

```

5655 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5656 {
5657   \group_begin:
5658   \tl_clear:N \l_@@_fill_tl
5659   \tl_clear:N \l_@@_opacity_tl
5660   \tl_clear:N \l_@@_color_tl
5661   \int_set:Nn \l_@@_key_nb_rows_int 1
5662   \dim_zero:N \l_@@_rounded_corners_dim
5663   \dim_zero:N \l_tmpa_dim
5664   \dim_zero:N \l_tmpb_dim
5665   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `fill` (or its alias `rowcolor`) has been used.

```

5666   \tl_if_empty:NF \l_@@_fill_tl
5667   {
5668     \@@_add_opacity_to_fill:
5669     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5670     {

```

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

```

5671     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5672     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5673     {
5674       \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5675       - *
5676     }
5677     { \dim_use:N \l_@@_rounded_corners_dim }
5678   }
5679 }
5680 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5681   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5682   {
5683     \@@_put_in_row_style:e
5684     {
5685       \@@_put_in_cell_after_hook:n
5686       {

```

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

```

5687       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5688       { \dim_use:N \l_tmpa_dim }
5689     }
5690   }
5691 }

```

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

```

5692   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5693   {
5694     \@@_put_in_row_style:e
5695     {
5696       \@@_put_in_cell_after_hook:n
5697       {
5698         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5699         { \dim_use:N \l_tmpb_dim }
5700       }
5701     }
5702   }

```

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

```

5703   \tl_if_empty:NF \l_@@_color_tl
5704   {
5705     \@@_put_in_row_style:e
5706     {
5707       \mode_leave_vertical:
5708       \@@_color:n { \l_@@_color_tl }
5709     }
5710   }

```

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

```

5711   \bool_if:NT \l_@@_bold_row_style_bool
5712   {
5713     \@@_put_in_row_style:n
5714     {
5715       \exp_not:n
5716       {
5717         \if_mode_math:
5718           $ % $
5719           \bfseries \boldmath
5720           $ % $
5721         \else:
5722           \bfseries \boldmath
5723         \fi:
5724       }
5725     }
5726   }
5727   \group_end:
5728   \g_@@_row_style_tl
5729   \ignorespaces
5730 }

```

The following commande must *not* be protected.

```

5731   \cs_new:Npn \@@_rounded_from_row:n #1
5732   {
5733     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5734 { \int_eval:n { #1 } - 1 }
5735 {
5736   \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5737   - \exp_not:n { \int_use:N \c@jCol }
5738 }
5739 { \dim_use:N \l_@@_rounded_corners_dim }
5740 }

```

## 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).

```

5741 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5742 {

```

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.

```

5743   \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`.

```

5744   \str_if_in:nnF { #1 } { !! }
5745   {
5746     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

5747     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5748   }
5749   \int_if_zero:nTF \l_tmpa_int

```

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

```

5750   {
5751     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5752     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5753   }

```

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

```

5754   { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5755   }
5756 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }

```

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

```

5757 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5758 {
5759   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5760   {

```

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

```

5761   \group_begin:
5762   \pgfsetcornersarced
5763   { \pgfpoint \l_@@_tab_rounded_corners_dim \l_@@_tab_rounded_corners_dim }

```

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.

```

5764   \bool_if:NTF \l_@@_hvlines_bool
5765   {
5766     \pgfpathrectanglecorners
5767     {
5768       \pgfpointadd
5769       { \@@_qpoint:n { row-1 } }
5770       { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5771     }
5772     {
5773       \pgfpointadd
5774       {
5775         \@@_qpoint:n
5776         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5777       }
5778       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5779     }
5780   }
5781   {
5782     \pgfpathrectanglecorners
5783     { \@@_qpoint:n { row-1 } }
5784     {
5785       \pgfpointadd
5786       {
5787         \@@_qpoint:n
5788         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5789       }
5790       { \pgfpoint \c_zero_dim \arrayrulewidth }
5791     }
5792   }
5793   \pgfusepath { clip }
5794   \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5795   }
5796 }

```

The command `\@@_actually_color:` will actually fill the rectangles, color by color, as specified in the sequence `\g_@@_colors_seq`.

```

5797 \cs_new_protected:Npn \@@_actually_color:
5798 {
5799   \pgfpicture
5800   \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.

```

5801   \@@_clip_with_rounded_corners:

```

We will now 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`).

```

5802   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5803   {
5804     \int_compare:nNnTF { ##1 } = 1
5805     {
5806       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5807       \use:c { g_@@_color _ 1 _tl }
5808       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5809     }
5810     {
5811       \pgfscope
5812       \@@_color_opacity: ##2
5813       \use:c { g_@@_color _ ##1 _tl }
5814       \tl_gclear:c { g_@@_color _ ##1 _tl }
5815       \pgfusepath { fill }
5816       \endpgfscope
5817     }
5818   }
5819   \endpgfpicture
5820 }

```

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

```

5821 \cs_new_protected:Npn \@@_color_opacity:
5822 {
5823   \peek_meaning:NTF [
5824     \@@_color_opacity:w
5825     { \@@_color_opacity:w [ ] }
5826   }

```

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.

```

5827 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5828 {
5829   \tl_clear:N \l_tmpa_tl
5830   \keys_set_known:nn { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

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

```

5831   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5832   \tl_if_empty:NTF \l_tmpb_tl
5833     \@declaredcolor
5834     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5835 }

```

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

```

5836 \keys_define:nn { nicematrix / color-opacity }
5837 {
5838   opacity .tl_set:N          = \l_tmpa_tl ,
5839   opacity .value_required:n = true
5840 }

```

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

```

5841 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5842 {
5843   \def \l_@@_rows_tl { #1 }
5844   \def \l_@@_cols_tl { #2 }
5845   \@@_cartesian_path:
5846 }

```

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

```
5847 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5848 {
5849   \tl_if_blank:nF { #2 }
5850   {
5851     \@@_add_to_colors_seq:en
5852     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5853     { \@@_cartesian_color:nn { #3 } { - } }
5854   }
5855 }
```

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

```
5856 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5857 {
5858   \tl_if_blank:nF { #2 }
5859   {
5860     \@@_add_to_colors_seq:en
5861     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5862     { \@@_cartesian_color:nn { - } { #3 } }
5863   }
5864 }
```

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

```
5865 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5866 {
5867   \tl_if_blank:nF { #2 }
5868   {
5869     \@@_add_to_colors_seq:en
5870     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5871     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5872   }
5873 }
```

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

```
5874 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5875 {
5876   \tl_if_blank:nF { #2 }
5877   {
5878     \@@_add_to_colors_seq:en
5879     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5880     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5881   }
5882 }
```

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

```
5883 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5884 {
5885   \@@_cut_on_hyphen:w #1 \q_stop
5886   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5887   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5888   \@@_cut_on_hyphen:w #2 \q_stop
5889   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5890   \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`.

```
5891 \@@_cartesian_path:n { #3 }
5892 }
```



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

```

5893 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5894 {
5895   \clist_map_inline:nn { #3 }
5896   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5897 }

5898 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5899 {
5900   \int_step_inline:nn \c@iRow
5901   {
5902     \int_step_inline:nn \c@jCol
5903     {
5904       \int_if_even:nTF { ####1 + ##1 }
5905       { \@@_cellcolor [ #1 ] { #2 } }
5906       { \@@_cellcolor [ #1 ] { #3 } }
5907       { ##1 - ####1 }
5908     }
5909   }
5910 }

```

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”.

```

5911 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5912 {
5913   \@@_rectanglecolor [ #1 ] { #2 }
5914   { 1 - 1 }
5915   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5916 }

5917 \keys_define:nn { nicematrix / rowcolors }
5918 {
5919   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5920   cols .tl_set:N = \l_@@_cols_tl ,
5921   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5922   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5923 }

```

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*.

```

5924 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5925 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5926   \seq_clear_new:N \l_@@_colors_seq
5927   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5928   \tl_clear_new:N \l_@@_cols_tl
5929   \tl_set:Nn \l_@@_cols_tl { - }
5930   \keys_set:nn { nicematrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

5931   \int_zero_new:N \l_@@_color_int
5932   \int_set:Nn \l_@@_color_int 1
5933   \bool_if:NT \l_@@_respect_blocks_bool
5934   {

```

We don't want to take into account a block which is entirely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5935      % modified 2026-02-18
5936      \seq_set_filter:NNn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5937      { \@@_not_in_exterior_p:nnnnn ##1 }
5938    }

```

#2 is the list of intervals of rows.

```

5939    \clist_map_inline:nn { #2 }
5940    {
5941      \tl_set:Nn \l_tmpa_tl { ##1 }
5942      \tl_if_in:NnTF \l_tmpa_tl { - }
5943      { \@@_cut_on_hyphen:w ##1 \q_stop }
5944      { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5945      \int_set:Nn \l_tmpa_int \l_tmpa_tl
5946      \int_set:Nn \l_@@_color_int
5947      { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5948      \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5949      \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5950      {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5951      \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).

```

5952      \bool_if:NT \l_@@_respect_blocks_bool
5953      {
5954        \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5955        { \@@_intersect_our_row_p:nnnnn #####1 }
5956        \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5957      }
5958      \tl_set:Nn \l_@@_rows_tl
5959      { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5960      \tl_set:Nn \l_@@_color_tl
5961      {
5962        \@@_color_index:n
5963        {
5964          \int_mod:nn
5965          { \l_@@_color_int - 1 }
5966          { \seq_count:N \l_@@_colors_seq }
5967          + 1
5968        }
5969      }
5970      \tl_if_empty:NF \l_@@_color_tl
5971      {
5972        \use:e
5973        {
5974          \@@_add_to_colors_seq:nn
5975          { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5976          { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5977        }
5978      }
5979      \int_incr:N \l_@@_color_int
5980      \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5981    }
5982  }
5983 }

```

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.

```
5984 \cs_new:Npn \@@_color_index:n #1
5985 {
```

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

```
5986 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5987 { \@@_color_index:n { #1 - 1 } }
5988 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5989 }
```

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

```
5990 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5991 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

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

```
5992 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5993 {
5994   \int_compare:nNtT { #3 } > \l_tmpb_int
5995   { \int_set:Nn \l_tmpb_int { #3 } }
5996 }
```

```
5997 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5998 {
5999   \int_if_zero:nTF { #4 }
6000   \prg_return_false:
6001   {
6002     \int_compare:nNtTF { #2 } > \c@jCol
6003     \prg_return_false:
6004     \prg_return_true:
6005   }
6006 }
```

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

```
6007 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6008 {
6009   \int_compare:nNtTF { #1 } > \l_tmpa_int
6010   \prg_return_false:
6011   {
6012     \int_compare:nNtTF \l_tmpa_int > { #3 }
6013     \prg_return_false:
6014     \prg_return_true:
6015   }
6016 }
```

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`).

```
6017 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6018 {
6019   \dim_compare:nNtTF { #1 } = \c_zero_dim
6020   {
6021     \bool_if:NtF \l_@@_nocolor_used_bool
6022     { \@@_cartesian_path_normal_ii: }
```

```

6023     {
6024         \clist_if_empty:NTF \l_@@_corners_cells_clist
6025         { \@@_cartesian_path_normal_i:n { #1 } }
6026         { \@@_cartesian_path_normal_ii: }
6027     }
6028 }
6029 { \@@_cartesian_path_normal_i:n { #1 } }
6030 }

```

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.

```

6031 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6032 {
6033     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6034     \clist_map_inline:Nn \l_@@_cols_tl
6035     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6036         \def \l_tmpa_tl { ##1 }
6037         \tl_if_in:NnTF \l_tmpa_tl { - }
6038         { \@@_cut_on_hyphen:w ##1 \q_stop }
6039         { \def \l_tmpb_tl { ##1 } }
6040         \tl_if_empty:NTF \l_tmpa_tl
6041         { \def \l_tmpa_tl { 1 } }
6042         {
6043             \str_if_eq:eeT \l_tmpa_tl { * }
6044             { \def \l_tmpa_tl { 1 } }
6045         }
6046         \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6047         { \@@_error:n { Invalid~col~number } }
6048         \tl_if_empty:NTF \l_tmpb_tl
6049         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6050         {
6051             \str_if_eq:eeT \l_tmpb_tl { * }
6052             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6053         }
6054         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6055         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6056         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6057         \@@_qpoint:n { col - \l_tmpa_tl }
6058         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6059         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6060         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6061         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6062         \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.

```

6063     \clist_map_inline:Nn \l_@@_rows_tl
6064     {
6065         \def \l_tmpa_tl { #####1 }
6066         \tl_if_in:NnTF \l_tmpa_tl { - }
6067         { \@@_cut_on_hyphen:w #####1 \q_stop }
6068         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6069         \tl_if_empty:NTF \l_tmpa_tl
6070         { \def \l_tmpa_tl { 1 } }
6071         {
6072             \str_if_eq:eeT \l_tmpa_tl { * }
6073             { \def \l_tmpa_tl { 1 } }
6074         }
6075         \tl_if_empty:NTF \l_tmpb_tl

```

```

6076         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6077         {
6078             \str_if_eq:eeT \l_tmpb_tl { * }
6079             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6080         }
6081         \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6082         { \@@_error:n { Invalid-row-number } }
6083         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6084         { \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`.

```

6085         \cs_if_exist:cF
6086         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6087         {
6088             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6089             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6090             \@@_qpoint:n { row - \l_tmpa_tl }
6091             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6092             \pgfpathrectanglecorners
6093             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6094             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6095         }
6096     }
6097 }
6098 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6099 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6100 {
6101     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6102     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6103     \clist_map_inline:Nn \l_@@_cols_tl
6104     {
6105         \@@_qpoint:n { col - ##1 }
6106         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6107         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6108         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6109         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6110         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6111     \clist_map_inline:Nn \l_@@_rows_tl
6112     {
6113         \@@_if_in_corner:nF { #####1 - ##1 }
6114         {
6115             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6116             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6117             \@@_qpoint:n { row - #####1 }
6118             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6119             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6120             {
6121                 \pgfpathrectanglecorners
6122                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6123                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6124             }
6125         }
6126     }
6127 }
6128 }

```

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`).

```
6129 \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

```
6130 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6131 {
6132   \bool_set_true:N \l_@@_nocolor_used_bool
6133   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6134   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
6135   \clist_map_inline:Nn \l_@@_rows_tl
6136   {
6137     \clist_map_inline:Nn \l_@@_cols_tl
6138     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6139   }
6140 }
```

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`.

```
6141 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6142 {
6143   \clist_set_eq:NN \l_tmpa_clist #1
6144   \clist_clear:N #1
6145   \clist_map_inline:Nn \l_tmpa_clist
6146   {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
6147     \def \l_tmpa_tl { ##1 }
6148     \tl_if_in:NnTF \l_tmpa_tl { - }
6149     { \@@_cut_on_hyphen:w ##1 \q_stop }
6150     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6151     \bool_lazy_or:nnT
6152     { \str_if_eq_p:ee \l_tmpa_tl { * } }
6153     { \tl_if_blank_p:o \l_tmpa_tl }
6154     { \def \l_tmpa_tl { 1 } }
6155     \bool_lazy_or:nnT
6156     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6157     { \tl_if_blank_p:o \l_tmpb_tl }
6158     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6159     \int_compare:nNnT \l_tmpb_tl > { #2 }
6160     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6161     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6162     { \clist_put_right:Nn #1 { ####1 } }
6163   }
6164 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
6165 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6166 {
6167   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6168   {
```

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`).

```
6169     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6170     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6171   }
6172   \ignorespaces
6173 }
```

```

6174 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6175 { \@@_error:n { cellcolor~in~Block } }
6176 % \end{macrocode}
6177 %
6178 % \begin{macrocode}
6179 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6180 { \@@_error:n { rowcolor~in~Block } }
6181 % \end{macrocode}
6182 %
6183 % \bigskip
6184 % The following command will be linked to |\rowcolor| in the tabular.
6185 % \begin{macrocode}
6186 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6187 {
6188   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6189   {
6190     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6191     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6192     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6193   }
6194   \ignorespaces
6195 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6196 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6197 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6198 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6199 {

```

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`.

```

6200   \seq_gclear:N \g_tmpa_seq
6201   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6202   { \@@_rowlistcolors_tabular:nnnn #1 }
6203   \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`.

```

6204   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6205   {
6206     { \int_use:N \c@iRow }
6207     { \exp_not:n { #1 } }
6208     { \exp_not:n { #2 } }
6209     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6210   }
6211   \ignorespaces
6212 }

```

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`).

```
6213 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6214 {
6215   \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).

```
6216   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6217   {
6218     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6219     {
6220       \@@_rowlistcolors
6221       [ \exp_not:n { #2 } ]
6222       { #1 - \int_eval:n { \c@iRow - 1 } }
6223       { \exp_not:n { #3 } }
6224       [ \exp_not:n { #4 } ]
6225     }
6226   }
6227 }
```

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.

```
6228 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6229 {
6230   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6231   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6232   \seq_gclear:N \g_@@_rowlistcolors_seq
6233 }

6234 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6235 {
6236   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6237   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6238 }
```

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.

```
6239 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6240 {
```

With the following line, we test whether the cell is the first one that we actually encounter in its column (don't forget that some rows may be incomplete and that an instruction `\multicolumn` may be used, leading to cells which are not "evaluated").

```
6241   \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6242   {
6243     \seq_gput_right:NV \g_@@_columncolor_seq \c@jCol
```

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).

```
6244   \tl_gput_left:Ne \g_@@_pre_code_before_tl
6245   {
6246     \exp_not:N \columncolor [ #1 ]
6247     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6248   }
6249 }
6250 }
```



```

6251 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6252 {
6253   \clist_map_inline:nn { #1 }
6254   {
6255     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6256     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6257     \columncolor { nocolor } { ##1 }
6258   }
6259 }
6260 \cs_new_protected:Npn \@@_EmptyRow:n #1
6261 {
6262   \clist_map_inline:nn { #1 }
6263   {
6264     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6265     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6266     \rowcolor { nocolor } { ##1 }
6267   }
6268 }

```

## 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`.

```

6269 \cs_new_eq:NN \OnlyMainNiceMatrix \use:n

```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```

6270 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6271 {
6272   \int_if_zero:nTF \l_@@_first_col_int
6273   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6274   {
6275     \int_if_zero:nTF \c@jCol
6276     {
6277       \int_compare:nNnF \c@iRow = { -1 }
6278       {
6279         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6280         { #1 }
6281       }
6282     }
6283     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6284   }
6285 }

```

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.

```

6286 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1

```

```

6287 {
6288   \int_if_zero:nF \c@iRow
6289   {
6290     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6291     {
6292       \int_compare:nNnT \c@jCol > \c_zero_int
6293       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6294     }
6295   }
6296 }

```

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`.

```

6297 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6298 {
6299   \IfPackageLoadedTF { tikz }
6300   {
6301     \IfPackageLoadedTF { booktabs }
6302     { #2 }
6303     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6304   }
6305   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6306 }
6307 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6308 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6309 \cs_new:Npn \@@_TopRule_i:
6310 {
6311   \noalign \bgroup
6312   \peek_meaning:NNTF [
6313   { \@@_TopRule_ii: }
6314   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6315 }
6316 \NewDocumentCommand \@@_TopRule_ii: { o }
6317 {
6318   \tl_gput_right:Ne \g_@@_rules_tl
6319   {
6320     \@@_draw_hrulerule:n
6321     {
6322       position = \int_eval:n { \c@iRow + 1 } ,
6323       tikz =
6324       {
6325         line-width = #1 ,
6326         yshift = 0.25 \arrayrulewidth ,
6327         shorten-< = - 0.5 \arrayrulewidth
6328       } ,
6329       total-width = #1
6330     }
6331   }
6332   \skip_vertical:n { \belowrulesep + #1 }
6333   \egroup
6334 }
6335 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6336 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6337 \cs_new:Npn \@@_BottomRule_i:
6338 {
6339   \noalign \bgroup
6340   \peek_meaning:NNTF [
6341   { \@@_BottomRule_ii: }
6342   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6343 }

```

```

6344 \NewDocumentCommand \@@_BottomRule_ii: { o }
6345 {
6346   \tl_gput_right:Ne \g_@@_rules_tl
6347   {
6348     \@@_draw_hrulerule:n
6349     {
6350       position = \int_eval:n { \c@iRow + 1 } ,
6351       tikz =
6352       {
6353         line-width = #1 ,
6354         yshift = 0.25 \arrayrulewidth ,
6355         shorten- < = - 0.5 \arrayrulewidth
6356       } ,
6357       total-width = #1 ,
6358     }
6359   }
6360   \skip_vertical:N \aboverulesep
6361   \@@_create_row_node_i:
6362   \skip_vertical:n { #1 }
6363   \egroup
6364 }

6365 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6366 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6367 \cs_new:Npn \@@_MidRule_i:
6368 {
6369   \noalign \bgroup
6370   \peek_meaning:NTF [
6371     { \@@_MidRule_ii: }
6372     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6373   ]

6374 \NewDocumentCommand \@@_MidRule_ii: { o }
6375 {
6376   \skip_vertical:N \aboverulesep
6377   \@@_create_row_node_i:
6378   \tl_gput_right:Ne \g_@@_rules_tl
6379   {
6380     \@@_draw_hrulerule:n
6381     {
6382       position = \int_eval:n { \c@iRow + 1 } ,
6383       tikz =
6384       {
6385         line-width = #1 ,
6386         yshift = 0.25 \arrayrulewidth ,
6387         shorten- < = - 0.5 \arrayrulewidth
6388       } ,
6389       total-width = #1 ,
6390     }
6391   }
6392   \skip_vertical:n { \belowrulesep + #1 }
6393   \egroup
6394 }

```

## General system for drawing rules

```

6395 \AtBeginDocument
6396 {
6397   \cs_new_protected:Npe \@@_draw_rules:
6398   {
6399     \c_@@_pgfortikzpicture_tl
6400     \@@_draw_rules_i:
6401     \c_@@_endpgfortikzpicture_tl

```

```

6402     }
6403 }
6404 \cs_new_protected:Npn \@@_draw_rules_i:
6405 {
6406   \bool_lazy_all:nT
6407   {
6408     { \clist_if_empty_p:N \l_@@_corners_clist }
6409     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6410     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6411     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6412     { ! \l_@@_fix_vertex_bool }
6413   }
6414   {
6415     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6416     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6417   }
6418   \pgfrememberpicturepositiononpagetrue
6419   \pgf@relevantforpicturesizefalse
6420   \pgfsetrectcap
6421   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6422   \CT@arc@
6423   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6424   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6425   \g_@@_rules_tl
6426 }

```

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in `\g_@@_rules_tl` a command `\@@_draw_vrule:n` or `\@@_draw_hrule: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 which is a kind of “internal set of keys”.

```

6427 \keys_define:nn { nicematrix / rules-after }
6428 {
6429   position .int_set:N = \l_@@_position_int ,
6430   start .int_set:N = \l_@@_start_int ,
6431   start .initial:n = 1 ,
6432   end .int_set:N = \l_@@_end_int ,
6433   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6434   dotted .code:n =
6435     \bool_set_true:N \l_@@_dotted_bool
6436     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6437     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6438   dotted .value_forbidden: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]`.

```

6439   color .code:n =
6440     \@@_set_CTarc:n { #1 }
6441     \CT@arc@
6442     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6443   color .value_required:n = true ,
6444   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6445   sep-color .value_required:n = true ,

```

Example for the key `total-width`:

```

\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = { line width = 1pt , dotted } ,
    total-width = 1 pt

```

```

    }
  }

6446   total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6447   unknown .code:n =
6448     \@@_unknown_key:nn
6449     { nicematrix / rules-after }
6450     { Unknown-key-for-a-rule } ,
6451   tikz .value_required:n = true
6452 }

```

If the user uses the key `tikz`, the rule (or more precisely: the different segments since a rule may be broken by blocks or others) will be drawn with TikZ.

```

6453 \AtBeginDocument
6454 {
6455   \IfPackageLoadedTF { tikz }
6456   {
6457     \keys_define:nn { nicematrix / rules-after }
6458     {
6459       tikz .code:n =
6460         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6461         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6462         \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6463       dashed .meta:n =
6464         {
6465           tikz = nicematrix / dashed ,
6466           total-width = \pgflinewidth
6467         }
6468     }
6469   }
6470   {
6471     \keys_define:nn { nicematrix / rules-after }
6472     { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6473   }
6474 }

6475 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6476 {
6477   \int_set:Nn \l_@@_multiplicity_int { #1 }
6478   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6479   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6480 }

6481 \AtBeginDocument
6482 {
6483   \IfPackageLoadedT { tikz }
6484   {
6485     \tikzset
6486     {
6487       nicematrix / dashed / .style =
6488       {
6489         dash-pattern = on-4~pt-off-2pt ,
6490         line-cap = butt
6491       }
6492     }
6493     \cs_if_exist:cT { tikz@library@decorations@loaded }
6494     { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6495   }
6496 }

```

## The vertical rules

`\@@_draw_vrule:n` and the socket `draw-vrule` will appear in `\@@_draw_key_vlines:n` and in the token list `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument #1 is a list of *key=value* pairs.

```
6497 \cs_new_protected:Npn \@@_draw_vrule:n #1
6498 {
```

The group is for the options.

```
6499 {
6500   \int_set_eq:NN \l_@@_end_int \c_iRow
6501   \keys_set:nn { nicematrix / rules-after } { #1 }
```

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).

```
6502   \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
6503   \socket_use:n { nicematrix / draw-vrule }
6504 }
6505 }
```

The socket “`nicematrix / draw-vrule`” will draw a vertical rule. That vertical rule may potentially be composed of several disconnected segments.

The plug `general` will break the vertical rule in several segments.

The plug `quick` will draw directly the vertical rule. That plug is used when we are sure that the whole rule must be drawn, that is to say when:

- there is no corners;
- there is no blocks;
- there is no implicit blocks created by the continuous dotted lines;
- there is no stroken blocks.

```
6506 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6507 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6508 {
6509   \group_begin:
```

`\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`.

```
6510   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
```

`\l_@@_x_initial_dim` will be the *x*-value of the rule to draw (used in all the plugs).

```
6511   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6512   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6513   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6514   \l_tmpa_tl
6515   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```
6516   \@@_test_v:
6517   \bool_if:NTF \g_tmpa_bool
6518   {
6519     \int_if_zero:nTF \l_@@_segment_start_int
```

We keep in memory that we have a rule to draw. `\l_@@_segment_start_int` will be the starting row of the rule that we will have to draw.

```
6520     { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6521     { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6522   }
6523   {
6524     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6525     {
6526       \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
```

The socket `vsegment` is defined below with its four plugs.

```

6527         \socket_use:n { nicematrix / vsegment }
6528         \int_zero:N \l_@@_segment_start_int
6529     }
6530 }
6531 }
6532 \int_compare:nNt \l_@@_segment_start_int > \c_zero_int
6533 {
6534     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6535     \socket_use:n { nicematrix / vsegment }
6536 }
6537 \group_end:
6538 }

6539 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6540 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6541 {
6542     \group_begin:
6543     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6544     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6545     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6546     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6547     \socket_use:n { nicematrix / vsegment }
6548     \group_end:
6549 }

```

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 `create-blocks-in-col` 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.

```

6550 \cs_new_protected:Npn \@@_test_v:
6551 {
6552     \bool_gset_true:N \g_tmpa_bool
6553     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6554     \bool_if:NT \g_tmpa_bool
6555     {
6556         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6557         { \@@_test_vline_in_block:nnnnn ##1 }
6558         \bool_if:NT \g_tmpa_bool
6559         {
6560             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6561             { \@@_test_vline_in_block:nnnnn ##1 }
6562             \bool_if:NT \g_tmpa_bool
6563             {
6564                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6565                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6566             }
6567         }
6568     }
6569 }

6570 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6571 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6572 {
6573     {
6574         \int_compare:nNtF \l_@@_position_int > \c@jCol
6575         { \bool_set_false:N \l_tmpa_bool }
6576         {
6577             \@@_test_h:
6578             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6579         }
6580         \int_compare:nNtF \l_@@_position_int = 1
6581         { \bool_gset_false:N \g_tmpa_bool }
6582         {

```

```

6583         \tl_set:Nn \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6584         \@@_test_h:
6585     }
6586     \bool_gset:Nn \g_tmpa_bool
6587     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6588 }
6589 \bool_if:NT \g_tmpa_bool
6590 {
6591     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6592     \socket_use:n { nicematrix / vsegment }
6593     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6594 }
6595 }

6596 \cs_new_protected:Npn \@@_test_in_corner_v:
6597 {
6598     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6599     {
6600         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6601         { \bool_set_false:N \g_tmpa_bool }
6602     }
6603     {
6604         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6605         {
6606             \int_compare:nNnTF \l_tmpb_tl = 1
6607             { \bool_set_false:N \g_tmpa_bool }
6608             {
6609                 \@@_if_in_corner:nT
6610                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6611                 { \bool_set_false:N \g_tmpa_bool }
6612             }
6613         }
6614     }
6615 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standard` for the simple rules.
- a plug `multiple` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug `dotted` for the rules with our own system of dotted rules;
- a plug `tikz` for the rules drawn with TikZ, including the key `dashed` of `custom-line`.

```

6616 \socket_new:nn { nicematrix / vsegment } { 0 }

```

In the following plug, the  $x$ -value for the line has been computed previously in `\l_@@_x_initial_dim`.

```

6617 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6618 {
6619     \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6620     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6621     \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6622     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6623     \pgfusepathqstroke
6624 }
6625 \socket_assign_plug:nn { nicematrix / vsegment } { standard }

```



```

6626 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6627 {
6628   \dim_add:Nn \l_@@_x_initial_dim
6629   {
6630     - 0.5 \l_@@_rule_width_after_dim
6631     +
6632     ( \arrayrulewidth * \l_@@_multiplicity_int
6633       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6634   }
6635   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6636   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6637   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6638   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6639   \bool_lazy_and:nnT
6640   { \cs_if_exist_p:N \CT@drsc@ }
6641   { ! \tl_if_blank_p:o \CT@drsc@ }
6642   {
6643     {
6644       \CT@drsc@
6645       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6646       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6647       \dim_set:Nn \l_@@_tmpd_dim
6648       {
6649         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6650         * ( \l_@@_multiplicity_int - 1 )
6651       }
6652       \pgfpathrectanglecorners
6653       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6654       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6655       \pgfusepath { fill }
6656     }
6657   }
6658   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6659   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6660   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6661   {
6662     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6663     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6664     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6665   }
6666   \pgfusepathqstroke
6667 }

6668 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6669 {
6670   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6671   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6672   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6673   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6674   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6675   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

The command `\@@_draw_line:` 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`

```

6676 \@@_draw_line:
6677 }

6678 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6679 {
6680 {

```

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).

```

6681 \tl_if_empty:NF \l_@@_rule_color_tl
6682 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6683 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6684 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6685 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6686 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6687 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6688 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6689 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6690 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6691 }
6692 }

```

The command `\@@_draw_key_vlines:` draws the horizontal rules specified by the key `vlines`. These rules are not drawn in the blocks (even the virtual blocks determined by commands such as `\Cdots`) and in the corners — if the key `corners` is used).

```

6693 \cs_new_protected:Npn \@@_draw_key_vlines:
6694 {
6695 {
6696 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6697 \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currrification in the following code.

```

6698 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6699 { \@@_lines_step_inline:n \c@jCol }
6700 { \clist_map_inline:Nn \l_@@_vlines_clist }
6701 {
6702 \int_set:Nn \l_@@_position_int { ##1 }
6703 \socket_use:n { nicematrix / draw-vrule }
6704 }
6705 }
6706 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6707 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6708 {
6709 \int_step_inline:nnn
6710 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6711 {
6712 \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6713 { #1 }
6714 { \int_eval:n { #1 + 1 } }
6715 }
6716 }

```

## The horizontal rules

`\@@_draw_hrule:n` and the socket `draw-hrule` will appear in `\@@_draw_key_hlines:n` and in the token rules `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6717 \cs_new_protected:Npn \@@_draw_hrule:n #1
6718 {
6719   {
6720     \int_set_eq:NN \l_@@_end_int \c@jCol
6721     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6722     \socket_use:nn { nicematrix / draw-hrule }
6723   }
6724 }
```

The socket “`nicematrix / draw-hrule`” will draw an horizontal rule. That horizontal rule may potentially be composed of several disconnected segments.

The plug `general` will break the vertical rule in several segments.

The plug `quick` will draw directly the vertical rule. That plug is used when we are sure that the whole rule must be drawn, that is to say when:

- there is no corners;
- there is no blocks;
- there is no implicit blocks created by the continuous dotted lines;
- there is no stroken blocks.

```

6725 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6726 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6727 {
6728   \group_begin:
```

`\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`.

```

6729   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
```

`\l_@@_y_initial_dim` will be the *y*-value of the rule to draw (used in all the plugs).

```

6730   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6731   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6732   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6733   \l_tmpb_tl
6734   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6735     \@@_test_h:
6736     \bool_if:NTF \g_tmpa_bool
6737     {
6738       \int_if_zero:nTF \l_@@_segment_start_int
```

We keep in memory that we have a segment to draw. `\l_@@_segment_start_int` will be the starting row of the rule that we will have to draw.

```

6739         { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6740         { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6741     }
6742   {
6743     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6744     {
6745       \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
```

The socket `hsegment` is defined below with its four plugs.

```

6746         \socket_use:n { nicematrix / hsegment }
6747         \int_zero:N \l_@@_segment_start_int
6748     }
6749 }
6750 }
6751 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6752 {
6753     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6754     \socket_use:n { nicematrix / hsegment }
6755 }
6756 \group_end:
6757 }
6758 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6759 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6760 {
6761     \group_begin:
6762     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6763     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6764     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6765     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6766     \socket_use:n { nicematrix / hsegment }
6767     \group_end:
6768 }

```

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 `create-blocks-in-col` 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.

```

6769 \cs_new_protected:Npn \@@_test_h:
6770 {
6771     \bool_gset_true:N \g_tmpa_bool
6772     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6773     \bool_if:NT \g_tmpa_bool
6774     {
6775         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6776         { \@@_test_hline_in_block:nnnnn ##1 }
6777         \bool_if:NT \g_tmpa_bool
6778         {
6779             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6780             { \@@_test_hline_in_block:nnnnn ##1 }
6781             \bool_if:NT \g_tmpa_bool
6782             {
6783                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6784                 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6785             }
6786         }
6787     }
6788 }
6789 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6790 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6791 {
6792     {
6793         \int_compare:nNnTF \l_@@_position_int > \c@iRow
6794         { \bool_set_false:N \l_tmpa_bool }
6795         {
6796             \@@_test_v:
6797             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6798         }
6799         \int_compare:nNnTF \l_@@_position_int = 1
6800         { \bool_gset_false:N \g_tmpa_bool }

```

```

6801     {
6802         \tl_set:Nn \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6803         \@@_test_v:
6804     }
6805     \bool_gset:Nn \g_tmpa_bool
6806     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6807 }
6808 \bool_if:NT \g_tmpa_bool
6809 {
6810     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6811     \socket_use:n { nicematrix / hsegment }
6812     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6813 }
6814 }

6815 \cs_new_protected:Npn \@@_test_in_corner_h:
6816 {
6817     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6818     {
6819         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6820         { \bool_set_false:N \g_tmpa_bool }
6821     }
6822     {
6823         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6824         {
6825             \int_compare:nNnTF \l_tmpa_tl = 1
6826             { \bool_set_false:N \g_tmpa_bool }
6827             {
6828                 \@@_if_in_corner:nT
6829                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6830                 { \bool_set_false:N \g_tmpa_bool }
6831             }
6832         }
6833     }
6834 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug `standard` for simple rules;
- a plug `multiple` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug `dotted` for the rules with our own system of dotted rules;
- a plug `tikz` for the rules drawn with TikZ, including the key `dashed` of `custom-line`.

```

6835 \socket_new:nn { nicematrix / hsegment } { 0 }

```

In the following plug, the  $y$ -value for the line has been computed previously in `\l_@@_y_initial_dim`.

```

6836 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6837 {
6838     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6839     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6840     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6841     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6842     \pgfusepathqstroke
6843 }
6844 \socket_assign_plug:nn { nicematrix / hsegment } { standard }

```

```

6845 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6846 {
6847   \dim_add:Nn \l_@@_y_initial_dim
6848   {
6849     - 0.5 \l_@@_rule_width_after_dim
6850     +
6851     ( \arrayrulewidth * \l_@@_multiplicity_int
6852       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6853   }
6854   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6855   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6856   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6857   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6858   \bool_lazy_and:nnT
6859   { \cs_if_exist_p:N \CT@drsc@ }
6860   { ! \tl_if_blank_p:o \CT@drsc@ }
6861   {
6862     {
6863       \CT@drsc@
6864       \dim_set:Nn \l_@@_tmpd_dim
6865       {
6866         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6867         * ( \l_@@_multiplicity_int - 1 )
6868       }
6869       \pgfpathrectanglecorners
6870       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6871       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6872       \pgfusepathqfill
6873     }
6874   }
6875   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6876   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6877   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6878   {
6879     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6880     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6881     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6882   }
6883   \pgfusepathqstroke
6884 }

```

Now, the case of a dotted line.

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

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}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

```

6885 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
6886 {
6887   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6888   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6889   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }

```

```

6890 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6891 \int_compare:nNtT \l_@@_segment_start_int = 1
6892 {
6893   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6894   \bool_if:NF \g_@@_delims_bool
6895   { \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.

```

6896   \tl_if_eq:NnF \g_@@_left_delim_tl (
6897     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6898   )
6899   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6900   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6901   \int_compare:nNtT \l_@@_segment_end_int = \c@jCol
6902   {
6903     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6904     \bool_if:NF \g_@@_delims_bool
6905     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6906     \tl_if_eq:NnF \g_@@_right_delim_tl )
6907     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6908   }

```

The command `\@@_draw_line:` 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`

```

6909 \@@_draw_line:
6910 }

```

```

6911 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6912 {
6913   {

```

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).

```

6914   \tl_if_empty:NF \l_@@_rule_color_tl
6915   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6916   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6917   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6918   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6919   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6920   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6921   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6922     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6923     -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6924   }
6925 }

```

The command `\@@_draw_key_hlines:` draws the horizontal rules specified by the key `hlines`. These rules are not drawn in the blocks (even the virtual blocks determined by commands such as `\Cdots`) and in the corners — if the key `corners` is used).

```

6926 \cs_new_protected:Npn \@@_draw_key_hlines:

```

```

6927 {
6928 {
6929 \dim_set_eq:Nn \l_@@_rule_width_after_dim \arrayrulewidth
6930 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6931 \str_if_eq:eeTF \l_@@_hlines_clist { all }
6932 { \@@_lines_step_inline:n \c@iRow }
6933 { \clist_map_inline:Nn \l_@@_hlines_clist }
6934 {
6935 \int_set:Nn \l_@@_position_int { ##1 }
6936 \socket_use:n { nicematrix / draw-hrule }
6937 }
6938 }
6939 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6940 \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.

```

6941 \cs_set:Npn \@@_Hline_i:n #1
6942 {
6943 \peek_remove_spaces:n
6944 {
6945 \peek_meaning:NTF \Hline
6946 { \@@_Hline_ii:nn { #1 + 1 } }
6947 { \@@_Hline_iii:n { #1 } }
6948 }
6949 }
6950 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6951 \cs_set:Npn \@@_Hline_iii:n #1
6952 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6953 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6954 {
6955 \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6956 \skip_vertical:N \l_@@_rule_width_before_dim
6957 \tl_gput_right:Ne \g_@@_rules_tl
6958 {

```

With the keys of `nicematrix / rules-after` we would write:

```

\@@_draw_hrule:n
{
multiplicity = #1 ,
position = \int_eval:n { \c@iRow + 1 } ,
total-width = \dim_use:N \l_@@_rule_width_before_dim ,
#2
}

```

We will use a version slightly more efficient:

```

6959 {
6960 \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6961 \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6962 \dim_set:Nn \l_@@_rule_width_after_dim
6963 { \dim_use:N \l_@@_rule_width_before_dim }
6964 \@@_draw_hrule:n { #2 }
6965 }
6966 }
6967 \egroup
6968 }

```



## 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`).

```

6969 \cs_new_protected:Npn \@@_custom_line:n #1
6970 {
6971   \str_clear:N \l_@@_letter_str
6972   \str_clear:N \l_@@_command_str
6973   \str_clear:N \l_@@_ccommand_str
6974   \tl_clear_new:N \l_@@_other_keys_tl
6975   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6976   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6977   {
6978     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6979     \str_set:Ne \l_@@_command_str
6980     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6981   }
6982   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6983   {
6984     \str_set:Ne \l_@@_ccommand_str
6985     { \str_tail:N \l_@@_ccommand_str }
6986     \str_set:Ne \l_@@_ccommand_str
6987     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6988   }

```

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.

```

6989   \bool_lazy_all:nTF
6990   {
6991     { \str_if_empty_p:N \l_@@_letter_str }
6992     { \str_if_empty_p:N \l_@@_command_str }
6993     { \str_if_empty_p:N \l_@@_ccommand_str }
6994   }
6995   { \@@_error:n { No~letter~and~no~command } }
6996   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6997 }

6998 \keys_define:nn { nicematrix / custom-line }
6999 {
7000   letter .str_set:N = \l_@@_letter_str ,
7001   letter .value_required:n = true ,
7002   command .str_set:N = \l_@@_command_str ,
7003   command .value_required:n = true ,
7004   ccommand .str_set:N = \l_@@_ccommand_str ,
7005   ccommand .value_required:n = true
7006 }

```

```

7007 \cs_new_protected:Npn \@@_custom_line_i:n #1
7008 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7009   \bool_set_false:N \l_@@_tikz_rule_bool
7010   \bool_set_false:N \l_@@_dotted_rule_bool
7011   \bool_set_false:N \l_@@_color_bool

```

```

7012 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7013 \bool_if:NT \l_@@_tikz_rule_bool
7014 {
7015     \IfPackageLoadedF { tikz }
7016     { \@@_error:n { tikz-in-custom-line-without-tikz } }
7017     \bool_if:NT \l_@@_color_bool
7018     { \@@_error:n { color-in-custom-line-with-tikz } }
7019 }
7020 \bool_if:NT \l_@@_dotted_rule_bool
7021 {
7022     \int_compare:nNnT \l_@@_multiplicity_int > 1
7023     { \@@_error:n { key-multiplicity-with-dotted } }
7024 }
7025 \str_if_empty:NF \l_@@_letter_str
7026 {
7027     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7028     { \@@_error:n { Several-letters } }
7029     {
7030         \tl_if_in:NoTF
7031         \c_@@_forbidden_letters_str
7032         \l_@@_letter_str
7033         { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
7034     }

```

During the analysis of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

7035         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7036         { \@@_v_custom_line:nn { #1 } }
7037     }
7038 }
7039 }
7040 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7041 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7042 }
7043 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7044 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
7045 \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-after}`). That's why the following set of keys has some keys which are no-op.

```

7046 \keys_define:nn { nicematrix / custom-line-bis }
7047 {
7048     multiplicity .int_set:N = \l_@@_multiplicity_int ,
7049     color .code:n = \bool_set_true:N \l_@@_color_bool ,
7050     color .value_required:n = true ,
7051     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7052     tikz .value_required:n = true ,
7053     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7054     dotted .value_forbidden:n = true ,
7055     dashed .meta:n =
7056     {
7057         tikz = nicematrix / dashed ,
7058         total-width = \pgflinewidth
7059     } ,
7060     total-width .code:n = { } ,
7061     total-width .value_required:n = true ,
7062     sep-color .code:n = { } ,
7063     sep-color .value_required:n = true ,
7064     unknown .code:n =
7065     \@@_unknown_key:nn

```

```

7066         { nicematrix / custom-line-bis }
7067         { Unknown-key-for-custom-line }
7068     }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7069 \bool_new:N \l_@@_dotted_rule_bool
7070 \bool_new:N \l_@@_tikz_rule_bool
7071 \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).

```

7072 \keys_define:nn { nicematrix / custom-line-width }
7073 {
7074     multiplicity .int_set:N = \l_@@_tmpc_int ,
7075     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7076     total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7077                     \bool_set_true:N \l_@@_total_width_bool ,
7078     total-width .value_required:n = true ,
7079     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7080 }

```

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 `\@@_draw_hruler:n` (which is in the internal `\CodeAfter`).

```

7081 \cs_new_protected:Npn \@@_h_custom_line:n #1
7082 {

```

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`).

```

7083     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7084     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7085 }

```

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 `\@@_draw_hruler:n` (which is in the internal `\CodeAfter`).

```

7086 \cs_new_protected:Npn \@@_c_custom_line:n #1
7087 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7088     \exp_args:Nc \DeclareExpandableDocumentCommand
7089     { nicematrix - \l_@@_ccommand_str }
7090     { 0 { } m }
7091     {
7092         \noalign
7093         {
7094             \@@_compute_rule_width:n { #1 , ##1 }
7095             \skip_vertical:n \l_@@_rule_width_before_dim
7096             \clist_map_inline:nn
7097             { ##2 }
7098             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7099         }
7100     }
7101     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7102 }

```

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*.

```

7103 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7104 {
7105   \tl_if_in:nnTF { #2 } { - }
7106   { \@@_cut_on_hyphen:w #2 \q_stop }
7107   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7108   \tl_gput_right:Ne \g_@@_rules_tl
7109   {
7110     \@@_draw_hrule:n
7111     {
7112       #1 ,
7113       start = \l_tmpa_tl ,
7114       end = \l_tmpb_tl ,
7115       position = \int_eval:n { \c@iRow + 1 } ,
7116       total-width = \dim_use:N \l_@@_rule_width_before_dim
7117     }
7118   }
7119 }

7120 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7121 {
7122   \bool_set_false:N \l_@@_tikz_rule_bool
7123   \bool_set_false:N \l_@@_total_width_bool
7124   \bool_set_false:N \l_@@_dotted_rule_bool
7125   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7126   \bool_if:NF \l_@@_total_width_bool
7127   {
7128     \bool_if:NTF \l_@@_dotted_rule_bool
7129     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7130     {
7131       \bool_if:NF \l_@@_tikz_rule_bool
7132       {
7133         \dim_set:Nn \l_@@_rule_width_before_dim
7134         {
7135           \arrayrulewidth * \l_@@_tmpc_int
7136           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7137         }
7138       }
7139     }
7140   }
7141 }

```

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

```

7142 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7143 {
7144   \str_if_eq:nnTF { #2 } { [ ] }
7145   { \@@_v_custom_line_i:nw { #1 } [ ] }
7146   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7147 }
7148 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7149 { \@@_v_custom_line:nn { #1 , #2 } }
7150 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7151 {
7152   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7153 \tl_gput_right:Ne \g_@@_array_preamble_tl
7154 {
7155   \exp_not:N !
7156   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7157 }
7158 \tl_gput_right:Ne \g_@@_rules_tl

```

```
7159 {
```

Here is a version with the keys of `rules-after`.

```
\@@_draw_vrule:n
{
  #2 ,
  position = \int_eval:n { \c@jCol + 1 } ,
  total-width = \dim_use:N \l_@@_rule_width_before_dim
}
```

However, you will use a slightly more efficient version.

```
7160 {
7161   \dim_set:Nn \l_@@_rule_width_after_dim
7162   { \dim_use:N \l_@@_rule_width_before_dim }
7163   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7164   \@@_draw_vrule:n { #2 }
7165 }
7166 }
7167 \@@_rec_preamble:n #1
7168 }

7169 \@@_custom_line:n
7170 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }
```

## The key `default-line`

```
7171 \keys_define:nn { nicematrix / default-line }
7172 {
7173   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7174   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7175   color .value_required:n = true ,
7176   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7177   dotted .value_forbidden:n = true ,
7178   total-width .code:n = { } ,
7179   total-width .value_required:n = true ,
7180   sep-color .code:n = { } ,
7181   sep-color .value_required:n = true ,
7182   unknown .code:n =
7183     \@@_unknown_key:nn
7184     { nicematrix / default-line }
7185     { Unknown-key-for~default-line }
7186 }

7187 \AtBeginDocument
7188 {
7189   \IfPackageLoadedT { tikz }
7190   {
7191     \keys_define:nn { nicematrix / default-line }
7192     {
7193       tikz .code:n =
7194         \bool_set_true:N \l_@@_tikz_rule_bool
7195         \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7196       tikz .value_required:n = true ,
7197       dashed .meta:n =
7198         {
7199           tikz = nicematrix / dashed ,
7200           total-width = \pgflinewidth
7201         }
7202     }
7203     \@@_custom_line:n
7204     {
7205       letter = ; ,
7206       command = \hdashedline ,
7207       ccommand = \cdashedline ,
```

```

7208         tikz = nicematrix / dashed ,
7209         total-width = \pgflinewidth
7210     }
7211 }
7212 }
7213 \cs_new_protected:Npn \@@_default_line:n #1
7214 {
7215     \bool_set_false:N \l_@@_tikz_rule_bool
7216     \bool_set_false:N \l_@@_dotted_rule_bool
7217     \bool_set_false:N \l_@@_color_bool
7218     \keys_set:nn { nicematrix / default-line } { #1 }
7219     \bool_if:NT \l_@@_tikz_rule_bool
7220     {
7221         \IfPackageLoadedF { tikz }
7222         { \@@_error:n { tikz-in-custom-line-without-tikz } }
7223         \bool_if:NT \l_@@_color_bool
7224         { \@@_error:n { color-in-custom-line-with-tikz } }
7225     }
7226     \bool_if:NT \l_@@_dotted_rule_bool
7227     {
7228         \int_compare:nNnT \l_@@_multiplicity_int > 1
7229         { \@@_error:n { key-multiplicity-with-dotted } }
7230     }
7231     \bool_if:NTF \l_@@_tikz_rule_bool
7232     {
7233         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7234         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7235     }
7236     {
7237         \bool_if:NTF \l_@@_dotted_rule_bool
7238         {
7239             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7240             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7241         }
7242         {
7243             \bool_if:NTF \l_@@_multiple_rule_bool
7244             {
7245                 \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7246                 \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7247             }
7248         }
7249     }
7250 }

```

## 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 `\g_tmpa_bool` is set to false.

```

7251 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7252 {
7253     \int_compare:nNnT \l_tmpa_tl > { #1 }
7254     {
7255         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7256         {
7257             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7258             {
7259                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7260                 { \bool_gset_false:N \g_tmpa_bool }
7261             }
7262         }
7263     }
7264 }

```

The same for vertical rules.

```

7265 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7266 {
7267   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7268   {
7269     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7270     {
7271       \int_compare:nNnT \l_tmpb_tl > { #2 }
7272       {
7273         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7274         { \bool_gset_false:N \g_tmpa_bool }
7275       }
7276     }
7277   }
7278 }

7279 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7280 {
7281   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7282   {
7283     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7284     {
7285       \int_compare:nNnTF \l_tmpa_tl = { #1 }
7286       { \bool_gset_false:N \g_tmpa_bool }
7287       {
7288         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7289         { \bool_gset_false:N \g_tmpa_bool }
7290       }
7291     }
7292   }
7293 }

7294 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7295 {
7296   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7297   {
7298     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7299     {
7300       \int_compare:nNnTF \l_tmpb_tl = { #2 }
7301       { \bool_gset_false:N \g_tmpa_bool }
7302       {
7303         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7304         { \bool_gset_false:N \g_tmpa_bool }
7305       }
7306     }
7307   }
7308 }

```

## 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.

```

7309 \cs_new_protected:Npn \@@_compute_corners:
7310 {
7311   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7312   { \@@_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`).

```

7313 \clist_clear:N \l_@@_corners_cells_clist
7314 \clist_map_inline:Nn \l_@@_corners_cells_clist
7315 {
7316   \str_case:nnF { ##1 }
7317   {
7318     { NW }
7319     { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7320     { NE }
7321     { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7322     { SW }
7323     { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7324     { SE }
7325     { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7326     { N }
7327     { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7328     { S }
7329     { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7330     { E }
7331     { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7332     { W }
7333     { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7334   }
7335   { \@@_error:nn { bad~corner } { ##1 } }
7336 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7337 \clist_if_empty:NF \l_@@_corners_cells_clist
7338 {

```

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.

```

7339 \tl_gput_right:Ne \g_@@_aux_tl
7340 {
7341   \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7342   { \l_@@_corners_cells_clist }
7343 }
7344 }
7345 }

```

```

7346 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7347 {
7348   \int_step_inline:nnn { #1 } { #3 }
7349   {
7350     \int_step_inline:nnn { #2 } { #4 }
7351     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
7352   }
7353 }

```

```

7354 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7355 {
7356   \cs_if_exist:cTF
7357   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7358   \prg_return_true:
7359   \prg_return_false:
7360 }

```

“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_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.

```
7361 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7362 {
```

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.

```
7363 \bool_set_false:N \l_tmpa_bool
7364 \int_zero_new:N \l_@@_last_empty_row_int
7365 \int_set:Nn \l_@@_last_empty_row_int { #1 }
7366 \int_step_inline:nnnn { #1 } { #3 } { #5 }
7367 {
7368   \bool_lazy_or:nnTF
7369   {
7370     \cs_if_exist_p:c
7371     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7372   }
7373   { \@@_if_in_block_p:nn { ##1 } { #2 } }
7374   { \bool_set_true:N \l_tmpa_bool }
7375   {
7376     \bool_if:NF \l_tmpa_bool
7377     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7378   }
7379 }
```

Now, you determine the last empty cell in the row of number 1.

```
7380 \bool_set_false:N \l_tmpa_bool
7381 \int_zero_new:N \l_@@_last_empty_column_int
7382 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7383 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7384 {
7385   \bool_lazy_or:nnTF
7386   {
7387     \cs_if_exist_p:c
7388     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7389   }
7390   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7391   { \bool_set_true:N \l_tmpa_bool }
7392   {
7393     \bool_if:NF \l_tmpa_bool
7394     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7395   }
7396 }
```

Now, we loop over the rows.

```
7397 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7398 {
```

We treat the row number `##1` with another loop.

```
7399 \bool_set_false:N \l_tmpa_bool
7400 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7401 {
7402   \bool_lazy_or:nnTF
7403   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7404   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
```

```

7405         { \bool_set_true:N \l_tmpa_bool }
7406     {
7407         \bool_if:NF \l_tmpa_bool
7408         {
7409             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7410             \clist_put_right:Nn
7411                 \l_@@_corners_cells_clist
7412                 { ##1 - #####1 }
7413             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7414         }
7415     }
7416 }
7417 }
7418 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7419 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7420 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:  
`\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...`

```

7421 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7422 {
7423     \int_step_inline:nn { #1 }
7424     {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7425         \bool_set_false:N \l_tmpa_bool
7426         \int_step_inline:nnnn { #2 } { #3 } { #4 }
7427         {
7428             \bool_lazy_or:nnTF
7429                 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 } }
7430                 { \@@_if_in_block_p:nn { #####1 } { ##1 } }
7431                 { \bool_set_true:N \l_tmpa_bool }
7432             {
7433                 \bool_if:NF \l_tmpa_bool
7434                 {
7435                     \clist_put_right:Nn
7436                         \l_@@_corners_cells_clist
7437                         { #####1 - ##1 }
7438                     \cs_set_nopar:cpn { @@ _ corner _ #####1 - ##1 } { }
7439                 }
7440             }
7441         }
7442     }
7443 }

```

```

7444 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7445 {
7446     \int_step_inline:nn { #1 }
7447     {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7448         \bool_set_false:N \l_tmpa_bool
7449         \int_step_inline:nnnn { #2 } { #3 } { #4 }
7450         {
7451             \bool_lazy_or:nnTF
7452                 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7453                 { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7454                 { \bool_set_true:N \l_tmpa_bool }
7455             {

```

```

7456         \bool_if:NF \l_tmpa_bool
7457         {
7458             \clist_put_right:Nn
7459             \l_@@_corners_cells_clist
7460             { ##1 - #####1 }
7461             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7462         }
7463     }
7464 }
7465 }
7466 }

```

## 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.

```

7467 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7468 \keys_define:nn { nicematrix / NiceMatrixBlock }
7469 {
7470     auto-columns-width .code:n =
7471     {
7472         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7473         \dim_gzero_new:N \g_@@_max_cell_width_dim
7474         \bool_set_true:N \l_@@_auto_columns_width_bool
7475     }
7476 }

7477 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
7478 {
7479     \int_gincr:N \g_@@_NiceMatrixBlock_int
7480     \dim_zero:N \l_@@_columns_width_dim
7481     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7482     \bool_if:NT \l_@@_block_auto_columns_width_bool
7483     {
7484         \cs_if_exist:cT
7485         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7486         {
7487             \dim_set:Nn \l_@@_columns_width_dim
7488             {
7489                 \use:c
7490                 { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7491             }
7492         }
7493     }
7494 }

```

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).

```

7495 {
7496     \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.

```

7497     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }

```

```

7498 {
7499   \bool_if:NT \l_@@_block_auto_columns_width_bool
7500   {
7501     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7502     \iow_shipout:Ne \@mainaux
7503     {
7504       \cs_gset:cpn
7505       { @@ _ 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.

```

7506       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7507     }
7508     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7509   }
7510 }
7511 \ignorespacesafterend
7512 }

```

## 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).

```

7513 \cs_new_protected:Npn \@@_create_extra_nodes:
7514 {
7515   \bool_if:nTF \l_@@_medium_nodes_bool
7516   {
7517     \bool_if:NTF \l_@@_no_cell_nodes_bool
7518     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7519     {
7520       \bool_if:NTF \l_@@_large_nodes_bool
7521       \@@_create_medium_and_large_nodes:
7522       \@@_create_medium_nodes:
7523     }
7524   }
7525   {
7526     \bool_if:NT \l_@@_large_nodes_bool
7527     {
7528       \bool_if:NTF \l_@@_no_cell_nodes_bool
7529       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7530       \@@_create_large_nodes:
7531     }
7532   }
7533 }

```

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`.

```

7534 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7535 {
7536   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7537   {
7538     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7539     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7540     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7541     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7542   }
7543   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7544   {
7545     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7546     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7547     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7548     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7549   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7550   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7551   {
7552     \int_step_variable:nnNn
7553     \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.

```

7554     {
7555       \cs_if_exist:cT
7556       { 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`.

```

7557     {
7558       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7559       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7560       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7561       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7562       {
7563         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7564         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7565       }

```

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`.

```

7566       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7567       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7568       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7569       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7570       {
7571         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7572         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
7573       }
7574     }
7575   }
7576 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7577   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7578   {
7579     \dim_compare:nNnT
7580     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim

```

```

7581     {
7582         \@@_qpoint:n { row - \@@_i: - base }
7583         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7584         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7585     }
7586 }
7587 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7588 {
7589     \dim_compare:nNnT
7590     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7591     {
7592         \@@_qpoint:n { col - \@@_j: }
7593         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7594         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7595     }
7596 }
7597 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7598 \cs_new_protected:Npn \@@_create_medium_nodes:
7599 {
7600     \pgfpicture
7601     \pgfrememberpicturepositiononpagetrue
7602     \pgf@relevantforpicturesizefalse
7603     \@@_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”.

```

7604     \tl_set:Nn \l_@@_suffix_tl { -medium }
7605     \@@_create_nodes:
7606     \endpgfpicture
7607 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>15</sup>. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7608 \cs_new_protected:Npn \@@_create_large_nodes:
7609 {
7610     \pgfpicture
7611     \pgfrememberpicturepositiononpagetrue
7612     \pgf@relevantforpicturesizefalse
7613     \@@_computations_for_medium_nodes:
7614     \@@_computations_for_large_nodes:
7615     \tl_set:Nn \l_@@_suffix_tl { -large }
7616     \@@_create_nodes:
7617     \endpgfpicture
7618 }
7619 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7620 {
7621     \pgfpicture
7622     \pgfrememberpicturepositiononpagetrue
7623     \pgf@relevantforpicturesizefalse
7624     \@@_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”.

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7625 \tl_set:Nn \l_@@_suffix_tl { - medium }
7626 \@@_create_nodes:
7627 \@@_computations_for_large_nodes:
7628 \tl_set:Nn \l_@@_suffix_tl { - large }
7629 \@@_create_nodes:
7630 \endpgfpicture
7631 }

```

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.

```

7632 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7633 {
7634   \int_set:Nn \l_@@_first_row_int 1
7635   \int_set:Nn \l_@@_first_col_int 1

```

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`.

```

7636   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7637   {
7638     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7639     {
7640       (
7641         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7642         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7643       )
7644       / 2
7645     }
7646     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7647     { l_@@_row _ \@@_i: _ min_dim }
7648   }
7649   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7650   {
7651     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7652     {
7653       (
7654         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7655         \dim_use:c
7656         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7657       )
7658       / 2
7659     }
7660     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7661     { l_@@_column _ \@@_j: _ max _ dim }
7662   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7663   \dim_sub:cn
7664   { l_@@_column _ 1 _ min _ dim }
7665   \l_@@_left_margin_dim
7666   \dim_add:cn
7667   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7668   \l_@@_right_margin_dim
7669 }

```

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).

```

7670 \cs_new_protected:Npn \@@_create_nodes:
7671 {

```

```

7672 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7673 {
7674   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7675   {

```

We draw the rectangular node for the cell ( $\backslash\@@_i-\backslash\@@_j$ ).

```

7676     \@@_pgf_rect_node:nnnnn
7677     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7678     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7679     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7680     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7681     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7682     \str_if_empty:NF \l_@@_name_str
7683     {
7684       \pgfnodealias
7685       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7686       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7687     }
7688   }
7689 }
7690 \int_step_inline:nn \c@iRow
7691 {
7692   \pgfnodealias
7693   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7694   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7695 }
7696 \int_step_inline:nn \c@jCol
7697 {
7698   \pgfnodealias
7699   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7700   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7701 }
7702 \pgfnodealias
7703 { \@@_env: - last - last \l_@@_suffix_tl }
7704 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol \l_@@_suffix_tl }

```

Now, we create the nodes for the cells of the  $\backslash\multicolumn$ . We recall that we have stored in  $\backslash\g_@@\_multicolumn\_cells\_seq$  the list of the cells where a  $\backslash\multicolumn\{n\}\{\dots\}\{\dots\}$  with  $n>1$  was issued and in  $\backslash\g_@@\_multicolumn\_sizes\_seq$  the correspondent values of  $n$ .

```

7705   \seq_map_pairwise_function:NNN
7706   \g_@@_multicolumn_cells_seq
7707   \g_@@_multicolumn_sizes_seq
7708   \@@_node_for_multicolumn:nn
7709 }

7710 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7711 {
7712   \cs_set_nopar:Npn \@@_i: { #1 }
7713   \cs_set_nopar:Npn \@@_j: { #2 }
7714 }

```

The command  $\backslash\@@\_node\_for\_multicolumn:nn$  takes two arguments. The first is the position of the cell where the command  $\backslash\multicolumn\{n\}\{\dots\}\{\dots\}$  was issued in the format  $i-j$  and the second is the value of  $n$  (the length of the “multi-cell”).

```

7715 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7716 {
7717   \@@_extract_coords_values: #1 \q_stop
7718   \@@_pgf_rect_node:nnnnn
7719   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7720   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7721   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7722   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
7723   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7724   \str_if_empty:NF \l_@@_name_str

```



```

7725 {
7726   \pgfnodealias
7727   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7728   { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7729 }
7730 }

```

## 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).

```

7731 \keys_define:nn { nicematrix / BlockFirstPass }
7732 {
7733   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7734     \bool_set_true:N \l_@@_p_block_bool ,
7735   j .value_forbidden:n = true ,
7736   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7737   l .value_forbidden:n = true ,
7738   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7739   r .value_forbidden:n = true ,
7740   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7741   c .value_forbidden:n = true ,
7742   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7743   L .value_forbidden:n = true ,
7744   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7745   R .value_forbidden:n = true ,
7746   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7747   C .value_forbidden:n = true ,
7748   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7749   t .value_forbidden:n = true ,
7750   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7751   T .value_forbidden:n = true ,
7752   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7753   b .value_forbidden:n = true ,
7754   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7755   B .value_forbidden:n = true ,
7756   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7757   m .value_forbidden:n = true ,
7758   v-center .meta:n = m ,
7759   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7760   p .value_forbidden:n = true ,
7761   respect-arraystretch .code:n =
7762     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7763   respect-arraystretch .value_forbidden:n = true ,

```

The following key is no longer documented in `nicematrix.tex` and `nicematrix-french.tex`. It should be considered as deprecated.

```

7764   color .code:n =
7765     \@@_color:n { #1 }
7766     \tl_set_rescan:Nnn
7767       \l_@@_draw_tl
7768       { \char_set_catcode_other:N ! }
7769       { #1 } ,
7770   color .value_required:n = true ,
7771 }

```

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.

```
7772 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```
7773 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7774 {
```

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).

```
7775   \tl_if_blank:nTF { #2 }
7776   { \@@_Block_ii:nnnnn 1 1 }
7777   {
7778     \tl_if_in:nnTF { #2 } { - }
7779     {
7780       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7781       \@@_Block_i_czech:w \@@_Block_i:w
7782       #2 \q_stop
7783     }
7784     {
7785       \@@_error:nn { Bad~argument~for~Block } { #2 }
7786       \@@_Block_ii:nnnnn 1 1
7787     }
7788   }
7789   { #1 } { #3 } { #4 }
7790   \ignorespaces
7791 }
```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```
7792 \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`.

```
7793 {
7794   \char_set_catcode_active:N -
7795   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7796 }
```

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.

```
7797 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7798 {
```

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).

```
7799   \bool_lazy_or:nnTF
7800   { \tl_if_blank_p:n { #1 } }
7801   { \str_if_eq_p:ee { * } { #1 } }
7802   { \int_set:Nn \l_tmpa_int { 100 } }
7803   { \int_set:Nn \l_tmpa_int { #1 } }
7804   \bool_lazy_or:nnTF
7805   { \tl_if_blank_p:n { #2 } }
7806   { \str_if_eq_p:ee { * } { #2 } }
7807   { \int_set:Nn \l_tmpb_int { 100 } }
7808   { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```

7809 \int_compare:nNnTF \l_tmpb_int = 1
7810 {
7811   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7812     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7813     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7814 }
7815 { \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.

```

7816 \keys_set_known:n { nicematrix / BlockFirstPass } { #3 }
7817 \tl_set:Nx \l_tmpa_tl
7818 {
7819   { \int_use:N \c@iRow }
7820   { \int_use:N \c@jCol }
7821   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7822   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7823 }

```

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).

```

7824 \bool_set_false:N \l_tmpa_bool
7825 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7826 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7827 \bool_case:nF
7828 {
7829   \l_tmpa_bool { \@@_Block_vii:eennn }
7830   \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7831 \l_@@_X_bool { \@@_Block_v:eennn }
7832 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7833 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7834 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7835 }
7836 { \@@_Block_v:eennn }
7837 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7838 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

`#1` is  $i$  (the number of rows of the block), `#2` is  $j$  (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the potential math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7839 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7840 {

```

```

7841 \int_gincr:N \g_@@_block_box_int
7842 \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7843 \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7844 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7845 {
7846   \tl_gput_right:Ne \g_@@_rules_tl
7847   {
7848     \@@_draw_diagbox:nnnnnn
7849     { \int_use:N \c@iRow }
7850     { \int_use:N \c@jCol }
7851     { \int_eval:n { \c@iRow + #1 - 1 } }
7852     { \int_eval:n { \c@jCol + #2 - 1 } }
7853     { \g_@@_row_style_tl \exp_not:n { ##1 } }
7854     { \g_@@_row_style_tl \exp_not:n { ##2 } }
7855   }
7856 }
7857 \box_gclear_new:c
7858 { 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!

```

7859 \hbox_gset:cn
7860 { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7861 {

```

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`).

```

7862 \tl_if_empty:NTF \l_@@_color_tl
7863 { \int_compare:nNnT { #2 } = 1 \set@color }
7864 { \@@_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`.

```

7865 \int_compare:nNnT { #1 } = 1
7866 {
7867   \int_if_zero:nTF \c@iRow
7868   {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7869         \cs_set_eq:NN \Block \@@_NullBlock:
7870         \l_@@_code_for_first_row_tl
7871     }
7872     {
7873         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7874         {
7875             \cs_set_eq:NN \Block \@@_NullBlock:
7876             \l_@@_code_for_last_row_tl
7877         }
7878     }
7879     \g_@@_row_style_tl
7880 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7881 \@@_reset_arraystretch:
7882 \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7883     #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.).

```

7884 \@@_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}`.

```

7885 \bool_if:NTF \l_@@_tabular_bool
7886 {
7887     \bool_lazy_all:nTF
7888     {
7889         { \int_compare_p:nNn { #2 } = 1 }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of  $-1$  cm.

```

7890         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7891         { ! \g_@@_rotate_bool }
7892     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7893     {
7894         \use:e
7895     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7896         \exp_not:N \begin { minipage }
7897         [ \str_lowercase:f \l_@@_vpos_block_str ]
7898         { \l_@@_col_width_dim }
7899         \str_case:on \l_@@_hpos_block_str
7900         { c \centering r \raggedleft l \raggedright }
7901     }
7902     #5
7903     \end { minipage }
7904 }

```

In the other cases, we use a `{tabular}`.

```

7905     {
7906         \use:e
7907     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7908         \exp_not:N \begin { tabular }
7909         [ \str_lowercase:f \l_@@_vpos_block_str ]
7910         { @ { } \l_@@_hpos_block_str @ { } }
7911     }
7912     #5
7913     \end { tabular }
7914 }
7915 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7916 {
7917     $ % $
7918     \bool_if:NT \l_@@_small_bool % 2026/04/05
7919     {
7920         \def \arraystretch { 0.47 }
7921         \dim_set:Nn \arraycolsep { 1.45 pt }
7922     }
7923     \use:e
7924     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7925         \exp_not:N \begin { array }
7926         [ \str_lowercase:f \l_@@_vpos_block_str ]
7927         { @ { } \l_@@_hpos_block_str @ { } }
7928     }
7929     \@@_tuning_key_small: % 2026/04/05
7930     #5
7931     \end { array }
7932     $ % $
7933 }
7934 }

```

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).

```

7935     \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.

```

7936     \int_compare:nNnT { #2 } = 1
7937     {
7938         \dim_gset:Nn \g_@@_blocks_wd_dim
7939         {
7940             \dim_max:nn
7941             \g_@@_blocks_wd_dim
7942             {
7943                 \box_wd:c
7944                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7945             }
7946         }
7947     }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7948     \int_compare:nNnT { #1 } = 1
7949     {
7950         \bool_lazy_any:nT
7951         {
7952             { \str_if_empty_p:N \l_@@_vpos_block_str }
7953             { \str_if_eq_p:ee t \l_@@_vpos_block_str }

```

```

7954         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7955     }
7956     \@@_adjust_blocks_ht_dp:
7957 }
7958 \seq_gput_right:Ne \g_@@_blocks_seq
7959 {
7960     \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.

```

7961     {
7962         \exp_not:n { #3 } ,
7963         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7964         \bool_if:NT \g_@@_rotate_bool
7965         {
7966             \bool_if:NTF \g_@@_rotate_c_bool
7967             { m }
7968             {
7969                 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7970                 { T }
7971             }
7972         }
7973     }
7974     {
7975         \box_use_drop:c
7976         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7977     }
7978 }
7979 \bool_set_false:N \g_@@_rotate_c_bool
7980 }

7981 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7982 {
7983     \dim_gset:Nn \g_@@_blocks_ht_dim
7984     {
7985         \dim_max:nn
7986         \g_@@_blocks_ht_dim
7987         {
7988             \box_ht:c
7989             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7990         }
7991     }
7992     \dim_gset:Nn \g_@@_blocks_dp_dim
7993     {
7994         \dim_max:nn
7995         \g_@@_blocks_dp_dim
7996         {
7997             \box_dp:c
7998             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7999         }
8000     }
8001 }

```

```

8002 \cs_new:Npn \@@_adjust_hpos_rotate:
8003 {
8004     \bool_if:NT \g_@@_rotate_bool
8005     {
8006         \str_set:Ne \l_@@_hpos_block_str
8007         {
8008             \bool_if:NTF \g_@@_rotate_c_bool

```

```

8009         c
8010       {
8011         \str_case:onF \l_@@_vpos_block_str
8012         { b l B l t r T r }
8013       {
8014         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
8015           r
8016           l
8017       }
8018     }
8019   }
8020 }
8021 }
8022 \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*.

```

8023 \cs_new_protected:Npn \@@_rotate_box_of_block:
8024 {
8025   \box_grotate:cn
8026   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8027   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8028   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8029   {
8030     \vbox_gset_top:cn
8031     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8032     {
8033       \skip_vertical:n { 0.8 ex }
8034       \box_use:c
8035       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8036     }
8037   }
8038   \bool_if:NT \g_@@_rotate_c_bool
8039   {
8040     \hbox_gset:cn
8041     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8042     {
8043       \IfFormatAtLeastTF { 2026-04-01 }
8044       {
8045         \vbox_center:n
8046         {
8047           \box_use:c
8048           { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8049         }
8050       }
8051       {
8052         $ % $
8053         \vcenter
8054         {
8055           \box_use:c
8056           { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8057         }
8058         $ % $
8059       }
8060     }
8061   }
8062 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key `p`). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).



#1 is  $i$  (the number of rows of the block), #2 is  $j$  (the number of columns of the block), #3 is the list of  $\text{key}=\text{values}$  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.

```

8063 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8064 {
8065   \seq_gput_right:Ne \g_@@_blocks_seq
8066   {
8067     \l_tmpa_tl
8068     { \exp_not:n { #3 } }
8069     {
8070       \bool_if:NTF \l_@@_tabular_bool
8071       {
8072         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8073       \@@_reset_arraystretch:
8074       \exp_not:n
8075       {
8076         \dim_zero:N \extrarowheight
8077         #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).

```

8078       \tag_if_active:T { \tag_stop:n { table } }
8079       \use:e
8080       {
8081         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8082         { @ { } \l_@@_hpos_block_str @ { } }
8083       }
8084       #5
8085       \end { tabular }
8086     }
8087   }
8088 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

8089 {
8090 {

```

The following will be no-op when `respect-arraystretch` is in force.

```

8091       \@@_reset_arraystretch:
8092       \exp_not:n
8093       {
8094         \dim_zero:N \extrarowheight
8095         #4
8096         $ % $
8097         \use:e
8098         {
8099           \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8100           { @ { } \l_@@_hpos_block_str @ { } }
8101         }
8102         \@@_tuning_key_small: % 2026/05/04
8103         #5
8104         \end { array }
8105         $ % $
8106       }
8107     }
8108   }
8109 }
8110 }
8111 }
8112 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8113 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8114 {
8115   \seq_gput_right:Ne \g_@@_blocks_seq
8116   {
8117     \l_tmpa_tl
8118     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8119     { { \exp_not:n { #4 #5 } } }
8120   }
8121 }
8122 \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`.

```

8123 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8124 {
8125   \seq_gput_right:Ne \g_@@_blocks_seq
8126   {
8127     \l_tmpa_tl
8128     { \exp_not:n { #3 } }
8129     { \exp_not:n { #4 #5 } }
8130   }
8131 }
8132 \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).

```

8133 \keys_define:nn { nicematrix / Block }
8134 {
8135   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8136   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8137   tikz .code:n =
8138     \IfPackageLoadedTF { tikz }
8139     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8140     { \@@_error:n { tikz~key~without~tikz } } ,
8141   tikz .value_required:n = true ,
8142   fill .code:n =
8143     \tl_set_rescan:Nnn
8144     \l_@@_fill_tl
8145     { \char_set_catcode_other:N ! }
8146     { #1 } ,
8147   fill .value_required:n = true ,

```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```

8148   opacity .tl_set:N = \l_@@_opacity_tl ,
8149   opacity .value_required:n = true ,
8150   draw .code:n =
8151     \tl_set_rescan:Nnn
8152     \l_@@_draw_tl
8153     { \char_set_catcode_other:N ! }
8154     { #1 } ,
8155   draw .default:n = default ,
8156   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8157   rounded-corners .default:n = 4 pt ,
8158   color .code:n =
8159     \@@_color:n { #1 }
8160     \tl_set_rescan:Nnn
8161     \l_@@_draw_tl

```

```

8162     { \char_set_catcode_other:N ! }
8163     { #1 } ,
8164     borders .clist_set:N = \l_@@_borders_clist ,
8165     borders .default:n = all , % added 2026/05/08
8166     hvlines .meta:n = { vlines , hlines } ,
8167     vlines .bool_set:N = \l_@@_vlines_block_bool ,
8168     hlines .bool_set:N = \l_@@_hlines_block_bool ,
8169     rules/width .dim_set:N = \arrayrulewidth ,
8170     rules/color .tl_set:N = \l_@@_rules_color_tl ,
8171     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8172     line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8173     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8174               \bool_set_true:N \l_@@_p_block_bool ,
8175     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8176     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8177     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8178     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8179               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8180     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8181               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8182     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8183               \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8184     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8185     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8186     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8187     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8188     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8189     m .value_forbidden:n = true ,
8190     v-center .meta:n = m ,
8191     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8192     p .value_forbidden:n = true ,
8193     name .str_set:N = \l_@@_block_name_str ,
8194     name .value_required:n = true ,
8195     respect-arraystretch .code:n =
8196       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8197     respect-arraystretch .value_forbidden:n = true ,
8198     transparent .bool_set:N = \l_@@_transparent_bool ,
8199     unknown .code:n =
8200       \@@_unknown_key:nn
8201         { nicematrix / Block }
8202         { Unknown-key-for-Block }
8203   }

```

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.

```

8204 \cs_new_protected:Npn \@@_draw_blocks:
8205 {
8206   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8207   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8208 }
8209 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8210 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8211   \int_zero:N \l_@@_last_row_int
8212   \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”).

```

8213 \int_compare:nNnTF { #3 } > { 98 }
8214 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8215 { \int_set:Nn \l_@@_last_row_int { #3 } }
8216 \int_compare:nNnTF { #4 } > { 98 }
8217 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8218 { \int_set:Nn \l_@@_last_col_int { #4 } }
8219 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8220 {
8221   \bool_lazy_and:nnTF
8222     \l_@@_preamble_bool
8223     {
8224       \int_compare_p:n
8225         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8226     }
8227     {
8228       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8229       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8230       \@@_msg_redirect_name:nn { columns-not-used } { none }
8231     }
8232     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8233 }
8234 {
8235   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8236   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8237   {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8238 \use:e
8239 {
8240   \@@_Block_v:nnnnnn
8241     { #1 }
8242     { #2 }
8243     { \int_use:N \l_@@_last_row_int }
8244     { \int_use:N \l_@@_last_col_int }
8245   }
8246   { #5 }
8247   { #6 }
8248 }
8249 }
8250 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of `key=value` options; #6 is the label (content) of the block.

```

8251 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8252 {

```

The group is for the keys.

```

8253 \group_begin:
8254 \int_compare:nNnT { #1 } = { #3 }
8255 { \str_set:Nn \l_@@_vpos_block_str { t } }
8256 \keys_set:nn { nicematrix / Block } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

8257 \bool_if:NT \l_@@_amp_in_blocks_bool
8258 { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

```

```

8259 \bool_lazy_and:nnT
8260   \l_@@_vlines_block_bool
8261   { ! \l_@@_ampersand_bool }
8262   {
8263     \tl_gput_right:Ne \g_@@_rules_tl
8264     {
8265       \@@_vlines_block:nnnnnn
8266       { \dim_use:N \arrayrulewidth }
8267       { \l_@@_rules_color_tl }
8268       { #1 } { #2 } { #3 } { #4 }
8269     }
8270   }
8271 \bool_if:NT \l_@@_hlines_block_bool
8272 {
8273   \tl_gput_right:Ne \g_@@_rules_tl
8274   {
8275     \@@_hlines_block:nnnnnn
8276     { \dim_use:N \arrayrulewidth }
8277     { \l_@@_rules_color_tl }
8278     { #1 } { #2 } { #3 } { #4 }
8279   }
8280 }

```

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).

```

8281 \bool_if:NF \l_@@_transparent_bool
8282 {
8283   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8284   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8285 }

8286 \tl_if_empty:NF \l_@@_draw_tl
8287 {
8288   \tl_gput_right:Nn \g_@@_rules_tl
8289   {
8290     \@@_stroke_block:nnnnn

```

#5 are the options

```

8291       { #5 } { #1 } { #2 } { #3 } { #4 }
8292     }
8293     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8294     { { #1 } { #2 } { #3 } { #4 } }
8295   }

8296 \clist_if_empty:NF \l_@@_borders_clist
8297 {
8298   \tl_gput_right:Nn \g_nicematrix_code_after_tl
8299   {
8300     \@@_stroke_borders_block:nnnnn
8301     { #5 } { #1 } { #2 } { #3 } { #4 }
8302   }
8303 }

8304 \tl_if_empty:NF \l_@@_fill_tl
8305 {
8306   \@@_add_opacity_to_fill:
8307   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8308   {
8309     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8310     { #1 - #2 }
8311     { #3 - #4 }
8312     { \dim_use:N \l_@@_rounded_corners_dim }
8313   }
8314 }

```

```

8315 \seq_if_empty:NF \l_@@_tikz_seq
8316 {
8317   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8318   {
8319     \@@_block_tikz:nnnnn
8320     { \seq_use:Nn \l_@@_tikz_seq { , } }
8321     { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8322   }
8323 }

8324 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8325 {
8326   \tl_gput_right:Ne \g_@@_rules_tl
8327   {
8328     \@@_draw_diagbox:nnnnnn
8329     { #1 } { #2 } { #3 } { #4 }
8330     { \exp_not:n { ##1 } }
8331     { \exp_not:n { ##2 } }
8332   }
8333 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\
& & two & \\
three & & four & five & \\
six & & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

8334 \pgfpicture
8335 \pgfrememberpicturepositiononpagetrue
8336 \pgf@relevantforpicturesizefalse
8337 \@@_qpoint:n { row - #1 }
8338 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8339 \@@_qpoint:n { col - #2 }
8340 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8341 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8342 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8343 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8344 \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.

```

8345 \@@_pgf_rect_node:nnnnn
8346 { \@@_env: - #1 - #2 - block }
8347 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8348 \str_if_empty:NF \l_@@_block_name_str
8349 {

```

```

8350 \pgfnodealias
8351 { \l_@@_env: - \l_@@_block_name_str }
8352 { \l_@@_env: - #1 - #2 - block }
8353 \str_if_empty:NF \l_@@_name_str
8354 {
8355   \pgfnodealias
8356   { \l_@@_name_str - \l_@@_block_name_str }
8357   { \l_@@_env: - #1 - #2 - block }
8358 }
8359 }

```

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.

```

8360 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8361 {
8362   \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.

```

8363 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8364 {

```

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.

```

8365   \cs_if_exist:cT
8366   { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
8367   {
8368     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8369     {
8370       \pgfpointanchor { \l_@@_env: - ##1 - #2 } { west }
8371       \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8372     }
8373   }
8374 }

```

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.

```

8375 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8376 {
8377   \l_@@_qpoint:n { col - #2 }
8378   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8379 }
8380 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8381 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8382 {
8383   \cs_if_exist:cT
8384   { pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8385   {
8386     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8387     {
8388       \pgfpointanchor
8389       { \l_@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8390       { east }
8391       \dim_set:Nn \l_@@_tmpd_dim
8392       { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8393     }
8394   }
8395 }
8396 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8397 {
8398   \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8399   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

```

8400     }
8401     \@_pgf_rect_node:nnnnn
8402     { \@_env: - #1 - #2 - block - short }
8403     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8404 }

```

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.

```

8405 \bool_if:NT \l_@@_medium_nodes_bool
8406 {
8407     \@_pgf_rect_node:nnn
8408     { \@_env: - #1 - #2 - block - medium }
8409     { \pgfpointanchor { \@_env: - #1 - #2 - medium } { north-west } }
8410     {
8411         \pgfpointanchor
8412         { \@_env:
8413             - \int_use:N \l_@@_last_row_int
8414             - \int_use:N \l_@@_last_col_int - medium
8415         }
8416         { south-east }
8417     }
8418 }
8419 \endpgfpicture
8420

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8421 \bool_if:NTF \l_@@_ampersand_bool
8422 {
8423     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8424     \int_zero_new:N \l_@@_split_int
8425     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell. We use locally counters that have another signification in the main environment (maybe we should change that).

```

8426 \int_set:Nn \l_@@_first_row_int { #1 }
8427 \int_set:Nn \l_@@_first_col_int { #2 }
8428 \int_set:Nn \l_@@_last_row_int { #3 }
8429 \int_set:Nn \l_@@_last_col_int { #4 }

8430 \pgfpicture
8431 \pgfrememberpicturerepositiononpagetrue
8432 \pgf@relevantforpicturesizefalse
8433 \@_qpoint:n { row - #1 }
8434 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8435 \@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8436 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8437 \@_qpoint:n { col - #2 }
8438 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8439 \@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8440 \dim_set:Nn \l_tmpb_dim
8441 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8442 \bool_lazy_or:nnT
8443 \l_@@_vlines_block_bool
8444 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8445 {
8446     \int_step_inline:nn { \l_@@_split_int - 1 }
8447     {
8448         \pgfpathmoveto
8449         {
8450             \pgfpoint
8451             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8452             \l_@@_tmpc_dim
8453         }

```



```

8454         \pgfpathlineto
8455         {
8456             \pgfpoint
8457             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8458             \l_@@_tmpd_dim
8459         }
8460         \CT@arc@
8461         \pgfsetlinewidth { 1.1 \arrayrulewidth }
8462         \pgfsetrectcap
8463         \pgfusepathqstroke
8464     }
8465 }
8466 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8467 \int_zero_new:N \l_@@_split_i_int
8468 \str_if_eq:eeTF \l_@@_vpos_block_str T
8469 {
8470     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8471     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8472 }
8473 {
8474     \str_if_eq:eeTF \l_@@_vpos_block_str B
8475     {
8476         \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8477         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8478     }
8479     {
8480         \bool_lazy_or:nnTF
8481         { \int_compare_p:nNn { #1 } = { #3 } }
8482         { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8483         {
8484             \@@_qpoint:n { row - #1 - base }
8485             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8486         }
8487         {
8488             \str_if_eq:eeTF \l_@@_vpos_block_str b
8489             {
8490                 \@@_qpoint:n { row - #3 - base }
8491                 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8492             }
8493             {
8494                 \@@_qpoint:n { #1 - #2 - block }
8495                 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8496             }
8497         }
8498     }
8499 }
8500 \int_step_inline:nn \l_@@_split_int
8501 {
8502     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8503     \int_set:Nn \l_@@_split_i_int { ##1 }
8504     \dim_set:Nn \col@sep
8505     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8506     \pgftransformshift
8507     {
8508         \pgfpoint
8509         {
8510             \l_tmpa_dim + ##1 \l_tmpb_dim -
8511             \str_case:on \l_@@_hpos_block_str
8512             {
8513                 l { \l_tmpb_dim + \col@sep }
8514                 c { 0.5 \l_tmpb_dim }
8515                 r { \col@sep }

```

```

8516         }
8517     }
8518     { \l_@@_tmpc_dim }
8519 }
8520 \pgfset { inner~sep = \c_zero_dim }
8521 \pgfnode
8522 { rectangle }
8523 {
8524     \str_if_eq:eeTF T \l_@@_vpos_block_str
8525     {
8526         \str_case:on \l_@@_hpos_block_str
8527         {
8528             l { north~west }
8529             c { north }
8530             r { north~east }
8531         }
8532     }
8533     {
8534         \str_if_eq:eeTF B \l_@@_vpos_block_str
8535         {
8536             \str_case:on \l_@@_hpos_block_str
8537             {
8538                 l { south~west }
8539                 c { south }
8540                 r { south~east }
8541             }
8542         }
8543         {
8544             \bool_lazy_any:nTF
8545             {
8546                 { \int_compare_p:nNn { #1 } = { #3 } }
8547                 { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8548                 { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8549             }
8550             {
8551                 \str_case:on \l_@@_hpos_block_str
8552                 {
8553                     l { base~west }
8554                     c { base }
8555                     r { base~east }
8556                 }
8557             }
8558             {
8559                 \str_case:on \l_@@_hpos_block_str
8560                 {
8561                     l { west }
8562                     c { center }
8563                     r { east }
8564                 }
8565             }
8566         }
8567     }
8568 }
8569 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8570 \group_end:
8571 }
8572 \endpgfpicture
8573 }

```

Now the case where there is no ampersand & in the content of the block.

```

8574 {
8575     \bool_if:NTF \l_@@_p_block_bool
8576     {

```

When the final user has used the key `p`, we have to compute the width.

```

8577 \pgfpicture
8578 \pgfrememberpicturepositiononpagetrue
8579 \pgf@relevantforpicturesizefalse
8580 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8581 {
8582   \@@_qpoint:n { col - #2 }
8583   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8584   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8585 }
8586 {
8587   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8588   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8589   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8590 }
8591 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8592 \endpgfpicture
8593 \hbox_set:Nn \l_@@_cell_box
8594 {
8595   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8596     { \g_tmpb_dim }
8597     \str_case:on \l_@@_hpos_block_str
8598       { c \centering r \raggedleft l \raggedright j { } }
8599     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8600     #6
8601     \end { minipage }
8602   }
8603 }
8604 {
8605   \hbox_set:Nn \l_@@_cell_box
8606   {
8607     \set@color
8608     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8609     #6
8610   }
8611 }
8612 \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.

```

8613 \pgfpicture
8614 \pgfrememberpicturepositiononpagetrue
8615 \pgf@relevantforpicturesizefalse
8616 \bool_lazy_any:nTF
8617 {
8618   { \str_if_empty_p:N \l_@@_vpos_block_str }
8619   { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8620   { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8621   { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8622 }
8623 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8624 \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`.

```

8625 \bool_if:nT \g_@@_last_col_found_bool
8626 {
8627   \int_compare:nNnT { #2 } = \g_@@_col_total_int
8628     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8629 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8630      \tl_set:Nc \l_tmpa_tl
8631      {
8632          \str_case:on \l_@@_vpos_block_str
8633          {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8634      { } {
8635          \str_case:on \l_@@_hpos_block_str
8636          {
8637              c { center }
8638              l { west }
8639              r { east }
8640              j { center }
8641          }
8642      }
8643      c {
8644          \str_case:on \l_@@_hpos_block_str
8645          {
8646              c { center }
8647              l { west }
8648              r { east }
8649              j { center }
8650          }
8651      }
8652      T {
8653          \str_case:on \l_@@_hpos_block_str
8654          {
8655              c { north }
8656              l { north-west }
8657              r { north-east }
8658              j { north }
8659          }
8660      }
8661      B {
8662          \str_case:on \l_@@_hpos_block_str
8663          {
8664              c { south }
8665              l { south-west }
8666              r { south-east }
8667              j { south }
8668          }
8669      }
8670      }
8671      }
8672      }
8673      }
8674      }
8675      \pgftransformshift
8676      {
8677          \pgfpointanchor
8678          {
8679              \@@_env: - #1 - #2 - block
8680              \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8681          }
8682          \l_tmpa_tl
8683      }
8684      \pgfset { inner~sep = \c_zero_dim }
8685      \pgfnode
8686      { rectangle }
8687      \l_tmpa_tl
8688      { \box_use_drop:N \l_@@_cell_box } { } { }

```

```
8689     }
```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```
8690     {
8691         \pgfextracty \l_tmpa_dim
8692         {
8693             \@@_qpoint:n
8694             {
8695                 row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8696                 - base
8697             }
8698         }
8699         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```
8700     \pgfpointanchor
8701     {
8702         \@@_env: - #1 - #2 - block
8703         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8704     }
8705     {
8706         \str_case:on \l_@@_hpos_block_str
8707         {
8708             c { center }
8709             l { west }
8710             r { east }
8711             j { center }
8712         }
8713     }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
8714     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8715     \pgfset { inner~sep = \c_zero_dim }
8716     \pgfnode
8717     { rectangle }
8718     {
8719         \str_case:on \l_@@_hpos_block_str
8720         {
8721             c { base }
8722             l { base~west }
8723             r { base~east }
8724             j { base }
8725         }
8726     }
8727     { \box_use_drop:N \l_@@_cell_box } { } { }
8728 }
8729 \endpgfpicture
8730 }
8731 \group_end:
8732 }
8733 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```
8734 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8735 {
8736     \tl_if_empty:NF \l_@@_opacity_tl
8737     {
8738         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8739             {
8740                 \tl_set:Ne \l_@@_fill_tl
8741                 {
```

```

8742         [ opacity = \l_@@_opacity_tl ,
8743         \tl_tail:o \l_@@_fill_tl
8744     ]
8745 }
8746 {
8747     \tl_set:Nx \l_@@_fill_tl
8748     { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8749 }
8750 }
8751 }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8752 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8753 {
8754     \group_begin:
8755     \tl_clear:N \l_@@_draw_tl
8756     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8757     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8758     \tl_if_empty:NF \l_@@_draw_tl
8759     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8760         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8761         \CT@arc@
8762         { \@@_color:o \l_@@_draw_tl }
8763     }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8764     \pgfsetcornersarced
8765     { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8766     \int_compare:nNnF { #2 } > \c@iRow
8767     {
8768         \int_compare:nNnF { #3 } > \c@jCol
8769         {
8770             \@@_qpoint:n { row - #2 }
8771             \dim_set_eq:NN \l_tmpb_dim \pgf@y
8772             \@@_qpoint:n { col - #3 }
8773             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8774             \@@_qpoint:n
8775             { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8776             \dim_set_eq:NN \l_tmpa_dim \pgf@y
8777             \@@_qpoint:n
8778             { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8779             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8780             \pgfpathrectanglecorners
8781             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8782             { \pgfpoint \pgf@x \l_tmpa_dim }
8783             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8784             \pgfusepathqstroke
8785             { \pgfusepath { stroke } }
8786         }
8787     }
8788     \group_end:
8789 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8790 \keys_define:nn { nicematrix / BlockStroke }
8791 {
8792     color .tl_set:N = \l_@@_draw_tl ,
8793     draw .code:n =
8794         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8795     draw .default:n = default ,

```

```

8796 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8797 rounded-corners .default:n = 4 pt ,
8798 rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8799 line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8800 }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The second is the color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8801 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8802 {
8803   \group_begin:

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```

8804   \dim_set:Nn \arrayrulewidth { #1 }
8805   \pgfsetlinewidth \arrayrulewidth % added 2026-03-28
8806   \@@_set_CTarc:n { #2 }
8807   \CT@arc@

```

We filter the list of blocks `\g_@@_pos_of_blocks_seq` in order to discard the blocks which encompass the current block (elsewhere, of course, the rules would not be drawn).

```

8808   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8809   {
8810     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8811     ||
8812     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8813     ||
8814     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8815     ||
8816     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8817   }
8818   \bool_if:NT \l_@@_fix_vertex_bool
8819   {
8820     \int_compare:nNnT { #4 } > 1
8821     {
8822       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8823       {
8824         { 1 }
8825         { 1 }
8826         { \int_use:N \c@iRow }
8827         { \int_eval:n { #4 -1 } }
8828         { }
8829       }
8830     }
8831     \int_compare:nNnT { #6 } < \c@jCol
8832     {
8833       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8834       {
8835         { 1 }
8836         { \int_eval:n { #6 + 1 } }
8837         { \int_use:N \c@iRow }
8838         { \int_use:N \c@jCol }
8839         { }
8840       }
8841     }
8842   }

```

```

8843 \int_set:Nn \l_@@_start_int { #3 }
8844 \int_set:Nn \l_@@_end_int { #5 }
8845 \int_step_inline:nnn { #4 } { #6 + 1 }
8846 {
8847   \int_set:Nn \l_@@_position_int { ##1 }
8848   \socket_use:n { nicematrix / draw-vrule }
8849 }
8850 \group_end:
8851 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8852 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8853 {
8854   \group_begin:

```

We are actually during the execution of the `\g_nicematrix_code_after_tl`. In that context `\arrayrulewidth` is the width of standard lines (we are drawing standard rules because, up to now, there is no way to draw the internal lines of a Block with a style of TikZ).

```

8855 \dim_set:Nn \arrayrulewidth { #1 }
8856 \pgfsetlinewidth \arrayrulewidth % added 2026/03/28
8857 \@@_set_CTarc:n { #2 }
8858 \CT@arc@

```

We filter the list of blocks `\g_@@_pos_of_blocks_seq` in order to discard the blocks which encompass the current block (elsewhere, of course, the rules would not be drawn).

```

8859 \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8860 {
8861   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8862   ||
8863   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8864   ||
8865   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8866   ||
8867   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8868 }
8869 \bool_if:NT \l_@@_fix_vertex_bool
8870 {
8871   \int_compare:nNnT { #3 } > 1
8872   {
8873     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8874     {
8875       { 1 }
8876       { 1 }
8877       { \int_eval:n { #3 - 1 } }
8878       { \int_use:N \c@jCol }
8879       { }
8880     }
8881   }
8882   \int_compare:nNnT { #5 } < \c@iRow
8883   {
8884     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8885     {
8886       { \int_eval:n { #5 + 1 } }
8887       { 1 }
8888       { \int_use:N \c@iRow }
8889       { \int_use:N \c@jCol }
8890       { }
8891     }
8892   }
8893 }

```



```

8894 \int_set:Nn \l_@@_start_int { #4 }
8895 \int_set:Nn \l_@@_end_int { #6 }
8896 \int_step_inline:nnn { #3 } { #5 + 1 }
8897 {
8898   \int_set:Nn \l_@@_position_int { ##1 }
8899   \socket_use:n { nicematrix / draw-hrule }
8900 }
8901 \group_end:
8902 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8903 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8904 {
8905   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8906   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8907
8908   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8909   { \@@_error:n { borders~forbidden } }
8910   {
8911     \tl_clear_new:N \l_@@_borders_tikz_tl
8912     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8913     \tl_set:Nn \l_@@_tmpc_tl { #2 }
8914     \tl_set:Nn \l_@@_tmpd_tl { #3 }
8915     \tl_set:Nc \l_tmpa_tl { \int_eval:n { #4 + 1 } }
8916     \tl_set:Nc \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8917     \@@_stroke_borders_block_i:
8918   }
8919 }
8920 \AtBeginDocument
8921 {
8922   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8923   {
8924     \c_@@_pgfortikzpicture_tl
8925     \@@_stroke_borders_block_ii:
8926     \c_@@_endpgfortikzpicture_tl
8927   }
8928 }
8929 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8930 {
8931   \pgfrememberpicturepositiononpagetrue
8932   \pgf@relevantforpicturesizefalse
8933   \CT@arc@
8934   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

If there is one specification of borders (right, left, top or bottom), we will raise the flag `\l_tmpa_bool` (and, if there isn't any specification of border, we will draw all the borders).

```

8935   \bool_set_false:N \l_tmpa_bool % added 2026/05/25
8936   \clist_if_in:NnT \l_@@_borders_clist { right }
8937   {
8938     \@@_stroke_vertical:n \l_tmpb_tl
8939     \bool_set_true:N \l_tmpa_bool
8940   }
8941   \clist_if_in:NnT \l_@@_borders_clist { left }
8942   {
8943     \@@_stroke_vertical:n \l_@@_tmpd_tl
8944     \bool_set_true:N \l_tmpa_bool
8945   }
8946   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8947   {
8948     \@@_stroke_horizontal:n \l_tmpa_tl
8949     \bool_set_true:N \l_tmpa_bool

```

```

8950     }
8951     \clist_if_in:NnT \l_@@_borders_clist { top }
8952     {
8953         \@@_stroke_horizontal:n \l_@@_tmpc_tl
8954         \bool_set_true:N \l_tmpa_bool
8955     }
8956     \bool_if:NF \l_tmpa_bool
8957     {
8958         \@@_stroke_vertical:n \l_tmpb_tl
8959         \@@_stroke_vertical:n \l_@@_tmpd_tl
8960         \@@_stroke_horizontal:n \l_tmpa_tl
8961         \@@_stroke_horizontal:n \l_@@_tmpc_tl
8962     }
8963 }
8964 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8965 {
8966     tikz .code:n =
8967         \cs_if_exist:NTF \tikzpicture
8968         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8969         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8970     tikz .value_required:n = true ,
8971     dashed .meta:n = { tikz = dashed } , % added 2026/05/25
8972     top .code:n = ,
8973     bottom .code:n = ,
8974     left .code:n = ,
8975     right .code:n = ,
8976     all .code:n = ,
8977     unknown .code:n = \@@_error:n { bad~border }
8978 }

```

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).

```

8979 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8980 {
8981     \@@_qpoint:n \l_@@_tmpc_tl
8982     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8983     \@@_qpoint:n \l_tmpa_tl
8984     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8985     \@@_qpoint:n { #1 }
8986     \tl_if_empty:NTF \l_@@_borders_tikz_tl
8987     {
8988         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8989         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8990         \pgfusepathqstroke
8991     }
8992     {
8993         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8994         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8995     }
8996 }

```

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).

```

8997 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8998 {
8999     \@@_qpoint:n \l_@@_tmpd_tl
9000     \clist_if_in:NnTF \l_@@_borders_clist { left }
9001     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9002     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9003     \@@_qpoint:n \l_tmpb_tl
9004     \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9005     \@@_qpoint:n { #1 }
9006     \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

9007 {
9008   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9009   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9010   \pgfusepathqstroke
9011 }
9012 {
9013   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9014   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9015 }
9016 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

9017 \keys_define:nn { nicematrix / BlockBorders }
9018 {
9019   borders .clist_set:N = \l_@@_borders_clist ,
9020   borders .default:n = all , % 2026/08/05
9021   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9022   rounded-corners .default:n = 4 pt ,
9023   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9024   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9025 }

```

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.

```

9026 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9027 {
9028   \begin { tikzpicture }
9029   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

9030   \clist_map_inline:nn { #1 }
9031   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9032   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9033   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9034   (
9035     [
9036       xshift = \dim_use:N \l_@@_offset_dim ,
9037       yshift = - \dim_use:N \l_@@_offset_dim
9038     ]
9039     #2 -| #3
9040   )
9041   rectangle
9042   (
9043     [
9044       xshift = - \dim_use:N \l_@@_offset_dim ,
9045       yshift = \dim_use:N \l_@@_offset_dim
9046     ]
9047     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9048   ) ;
9049   }
9050   \end { tikzpicture }
9051 }
9052 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

9053 \keys_define:nn { nicematrix / SpecialOffset }
9054 {
9055     offset .dim_set:N = \l_@@_offset_dim ,
9056 }

```

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.

```

9057 \cs_new_protected:Npn \@@_NullBlock:
9058 { \@@_collect_options:n { \@@_NullBlock_i: } }
9059 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9060 { }

```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (&). Of course, `&-in-blocks` must be in force.

```

9061 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9062 {
9063     \tl_gput_right:Nx \g_@@_pre_code_before_tl
9064     {

```

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`).

```

9065         \@@_subcellcolor:nnnnnnn
9066         {
9067             \tl_if_blank:nTF { #1 }
9068             { { \exp_not:n { #2 } } }
9069             { [ #1 ] { \exp_not:n { #2 } } }
9070         }
9071         { \int_use:N \l_@@_first_row_int } % first row of the block
9072         { \int_use:N \l_@@_first_col_int } % first column of the block
9073         { \int_use:N \l_@@_last_row_int } % last row of the block
9074         { \int_use:N \l_@@_last_col_int } % last column of the block
9075         { \int_use:N \l_@@_split_int }
9076         { \int_use:N \l_@@_split_i_int }
9077     }
9078     \ignorespaces
9079 }

9080 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9081 {
9082     \@@_color_opacity: #1
9083     \pgfpicture
9084     \pgf@relevantforpicturesizefalse
9085     \@@_qpoint:n { col - #3 }
9086     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9087     \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9088     \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9089     \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9090     \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9091     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9092     \@@_qpoint:n { row - #2 }
9093     \pgfpathrectanglecorners
9094     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9095     { \pgfpoint \l_tmpb_dim \pgf@y }
9096     \pgfusepathqfill
9097     \endpgfpicture
9098 }

```

## 27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

9099 \keys_define:nn { nicematrix / Auto }
9100 {
9101   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9102   columns-type .value_required:n = true ,
9103   l .meta:n = { columns-type = l } ,
9104   r .meta:n = { columns-type = r } ,
9105   c .meta:n = { columns-type = c } ,
9106   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9107   delimiters / color .value_required:n = true ,
9108   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9109   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9110   delimiters .value_required:n = true ,
9111   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9112   rounded-corners .default:n = 4 pt
9113 }

9114 \NewDocumentCommand \AutoNiceMatrixWithDelims
9115 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9116 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

9117 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9118 {

```

The group is for the protection of the keys.

```

9119   \group_begin:
9120   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9121   \use:e
9122   {
9123     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9124     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9125     [ \exp_not:o \l_tmpa_tl ]
9126   }
9127   \int_if_zero:nT \l_@@_first_row_int
9128   {
9129     \int_if_zero:nT \l_@@_first_col_int { & }
9130     \prg_replicate:nn { #4 - 1 } { & }
9131     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9132   }
9133   \prg_replicate:nn { #3 }
9134   {
9135     \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).

```

9136     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9137     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9138   }
9139   \int_compare:nNnT \l_@@_last_row_int > { -2 }
9140   {
9141     \int_if_zero:nT \l_@@_first_col_int { & }
9142     \prg_replicate:nn { #4 - 1 } { & }
9143     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9144   }
9145   \end { NiceArrayWithDelims }
9146   \group_end:
9147 }

9148 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9149 {
9150   \cs_set_protected:cpn { #1 AutoNiceMatrix }

```

```

9151     {
9152         \bool_gset_true:N \g_@@_delims_bool
9153         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9154         \AutoNiceMatrixWithDelims { #2 } { #3 }
9155     }
9156 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

9157 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
9158 {
9159     \group_begin:
9160     \bool_gset_false:N \g_@@_delims_bool
9161     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9162     \group_end:
9163 }

```

## 28 The redefinition of the command `\dotfill`

```

9164 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9165 \cs_new_protected:Npn \@@_dotfill:
9166 {

```

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}`).

```

9167     \@@_old_dotfill:
9168     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9169 }

```

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`.

```

9170 \cs_new_protected:Npn \@@_dotfill_i:
9171 {
9172     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9173     { \@@_old_dotfill: }
9174 }

```

## 29 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.

```

9175 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9176 {
9177     \tl_gput_right:Ne \g_@@_rules_tl
9178     {
9179         \@@_draw_diagbox:nnnnnn
9180         { \int_use:N \c@iRow }
9181         { \int_use:N \c@jCol }
9182         { \int_use:N \c@iRow }
9183         { \int_use:N \c@jCol }

```

The expansion done on `\g_@@_row_style_tl` will result in the fact that you take into account the current number of row and number of column.

```

9184         { \g_@@_row_style_tl \exp_not:n { #1 } }
9185         { \g_@@_row_style_tl \exp_not:n { #2 } }
9186     }

```

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`.

```

9187   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9188   {
9189     { \int_use:N \c@iRow }
9190     { \int_use:N \c@jCol }
9191     { \int_use:N \c@iRow }
9192     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9193     { }
9194   }
9195 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_draw_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

9196 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9197 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9198   \begin { pgfscope }
9199   \@@_qpoint:n { row - #1 }
9200   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9201   \@@_qpoint:n { col - #2 }
9202   \dim_set_eq:NN \l_tmpb_dim \pgf@x
9203   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9204   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9205   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9206   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9207   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9208   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9209   {

```

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.

```

9210   \CT@arc@
9211   \pgfsetroundcap
9212   \pgfusepathqstroke
9213 }
9214 \pgfset { inner~sep = 1 pt }
9215 \pgfscope
9216 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9217 \pgfnode { rectangle } { south~west }
9218 {
9219   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

9220   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9221   \end { minipage }
9222 }
9223 { }
9224 { }
9225 \endpgfscope
9226 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9227 \pgfnode { rectangle } { north~east }
9228 {
9229   \begin { minipage } { 20 cm }
9230   \raggedleft
9231   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9232   \end { minipage }
9233 }
9234 { }

```

```

9235     { }
9236   \end { pgfscope }
9237 }

```

## 30 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 90.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9238 \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 `\`.

```

9239 \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`.

```

9240 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9241 {
9242   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9243   \@@_CodeAfter_iv:n
9244 }

```

We catch the argument of the command `\end` (in `#1`).

```

9245 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9246 {

```

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.

```

9247   \str_if_eq:eeTF \@currenvir { #1 }
9248   { \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`.

```

9249   {
9250     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9251     \@@_CodeAfter_ii:n
9252   }
9253 }

```

## 31 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.



```

9254 \cs_new_protected:Npn \l_@@_delimiter:nnn #1 #2 #3
9255 {
9256   \pgfpicture
9257   \pgfrememberpicturepositiononpagetrue
9258   \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.

```

9259   \l_@@_qpoint:n { row - 1 }
9260   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9261   \l_@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9262   \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).

```

9263   \bool_if:nTF { #3 }
9264   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9265   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9266   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9267   {
9268     \cs_if_exist:cT
9269     { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
9270     {
9271       \pgfpointanchor
9272       { \l_@@_env: - ##1 - #2 }
9273       { \bool_if:nTF { #3 } { west } { east } }
9274       \dim_set:Nn \l_tmpa_dim
9275       {
9276         \bool_if:nTF { #3 }
9277         \dim_min:nn
9278         \dim_max:nn
9279         \l_tmpa_dim
9280         \pgf@x
9281       }
9282     }
9283   }

```

Now we can put the delimiter with a node of PGF.

```

9284   \pgfset { inner~sep = \c_zero_dim }
9285   \dim_zero:N \nulldelimiterspace
9286   \pgftransformshift
9287   {
9288     \pgfpoint
9289     \l_tmpa_dim
9290     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9291   }
9292   \pgfnode
9293   { rectangle }
9294   { \bool_if:nTF { #3 } { east } { west } }
9295   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9296   \nullfont
9297   $ % $
9298   \l_@@_color:o \l_@@_delimiters_color_tl
9299   \bool_if:nTF { #3 } { \left #1 } { \left . }
9300   \vcenter
9301   {
9302     \nullfont
9303     \hrule \@height
9304     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9305     \@depth \c_zero_dim
9306     \@width \c_zero_dim
9307   }

```

```

9308     \bool_if:nTF { #3 } { \right . } { \right #1 }
9309     $ % $
9310 }
9311 { }
9312 { }
9313 \endpgfpicture
9314 }

```

## 32 The command \SubMatrix

```

9315 \keys_define:nn { nicematrix / sub-matrix }
9316 {
9317     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9318     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9319     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9320     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9321     xshift .value_required:n = true ,
9322     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9323     delimiters / color .value_required:n = true ,
9324     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9325     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9326     hlines .default:n = all ,
9327     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9328     vlines .default:n = all ,
9329     hvlines .meta:n = { hlines, vlines } ,
9330     hvlines .value_forbidden:n = true
9331 }
9332 \keys_define:nn { nicematrix }
9333 {
9334     SubMatrix .inherit:n = nicematrix / sub-matrix ,
9335     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9336     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9337     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9338 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

9339 \keys_define:nn { nicematrix / SubMatrix }
9340 {
9341     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9342     delimiters / color .value_required:n = true ,
9343     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9344     hlines .default:n = all ,
9345     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9346     vlines .default:n = all ,
9347     hvlines .meta:n = { hlines, vlines } ,
9348     hvlines .value_forbidden:n = true ,
9349     name .code:n =
9350     \tl_if_empty:nTF { #1 }
9351     { \@@_error:n { Invalid-name } }
9352     {
9353         \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9354         {
9355             \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9356             { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9357             {
9358                 \str_set:Nn \l_@@_submatrix_name_str { #1 }
9359                 \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9360             }
9361         }
9362     }

```

```

9362         { \@@_error:n { Invalid-name } }
9363     } ,
9364     name .value_required:n = true ,
9365     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9366     rules .value_required:n = true ,
9367     code .tl_set:N = \l_@@_code_tl ,
9368     code .value_required:n = true ,
9369     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9370 }

9371 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9372 {
9373     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9374     {
9375         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9376         [
9377             delimiters / color = \l_@@_delimiters_color_tl ,
9378             hlines = \l_@@_submatrix_hlines_clist ,
9379             vlines = \l_@@_submatrix_vlines_clist ,
9380             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9381             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9382             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9383             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9384             #5
9385         ]
9386     }
9387     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9388     \ignorespaces
9389 }

9390 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9391 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9392 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9393 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9394 {
9395     \seq_gput_right:Ne \g_@@_submatrix_seq
9396     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9397         { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9398         { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9399         { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9400         { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9401     }
9402 }

```

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).

```

9403 \NewDocumentCommand \@@_compute_i_j:nn
9404 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9405 { \@@_compute_i_j:nnnn #1 #2 }

9406 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9407 {
9408     \def \l_@@_first_i_tl { #1 }
9409     \def \l_@@_first_j_tl { #2 }
9410     \def \l_@@_last_i_tl { #3 }
9411     \def \l_@@_last_j_tl { #4 }
9412     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9413     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9414     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9415     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9416     \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

9417     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9418     \tl_if_eq:NnT \l_@@_last_j_tl { last }
9419     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9420 }

```

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`.

```

9421 \AtBeginDocument
9422 {
9423     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9424     \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9425     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9426 }
9427 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9428 {
9429     \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9430     \@@_compute_i_j:nn { #2 } { #3 }
9431     \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
9432     { \def \arraystretch { 1 } }
9433     \bool_lazy_or:nnTF
9434     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9435     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9436     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9437     {
9438         \str_clear_new:N \l_@@_submatrix_name_str
9439         \keys_set:nn { nicematrix / SubMatrix } { #5 }
9440         \pgfpicture
9441         \pgfrememberpicturepositiononpagetrue
9442         \pgf@relevantforpicturesizefalse
9443         \pgfset { inner~sep = \c_zero_dim }
9444         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9445         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9446     \bool_if:NTF \l_@@_submatrix_slim_bool
9447     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9448     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9449     {
9450         \cs_if_exist:cT
9451         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9452         {
9453             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9454             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9455             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9456         }
9457         \cs_if_exist:cT

```

```

9458         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9459         {
9460             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9461             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9462             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9463         }
9464     }
9465     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9466     { \@@_impossible_delimiter:n { left } }
9467     {
9468         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9469         { \@@_impossible_delimiter:n { right } }
9470         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9471     }
9472     \endpgfpicture
9473 }
9474 \group_end:
9475 \ignorespaces
9476 }

```

The argument of the following command will be provided by curryfication.

```

9477 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9478 {
9479     \bool_if:NTF \l_@@_no_cell_nodes_bool
9480     { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9481     { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9482 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9483 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9484 {
9485     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9486     \dim_set:Nn \l_@@_y_initial_dim
9487     {
9488         \fp_to_dim:n
9489         {
9490             \pgf@y
9491             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9492         }
9493     }
9494     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9495     \dim_set:Nn \l_@@_y_final_dim
9496     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9497     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9498     {
9499         \cs_if_exist:cT
9500         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9501         {
9502             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9503             \dim_set:Nn \l_@@_y_initial_dim
9504             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9505         }
9506         \cs_if_exist:cT
9507         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9508         {
9509             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9510             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9511             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9512         }
9513     }
9514     \dim_set:Nn \l_tmpa_dim
9515     {

```

```

9516         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9517         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9518     }
9519     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9520     \group_begin:
9521     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9522     \@@_set_CTarc:o \l_@@_rules_color_tl
9523     \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`.

```

9524     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9525     {
9526         \int_compare:nNt \l_@@_first_j_tl < { ##1 }
9527         {
9528             \int_compare:nNt
9529             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9530             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9531                 \@@_qpoint:n { col - ##1 }
9532                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9533                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9534                 \pgfusepathqstroke
9535             }
9536         }
9537     }

```

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.

```

9538     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9539     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9540     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9541     {
9542         \bool_lazy_and:nnTF
9543         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9544         {
9545             \int_compare_p:nNn
9546             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9547         {
9548             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9549             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9550             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9551             \pgfusepathqstroke
9552         }
9553         { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
9554     }

```

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.

```

9555     \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9556     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9557     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9558     {
9559         \bool_lazy_and:nnTF
9560         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9561         {
9562             \int_compare_p:nNn
9563             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9564         {
9565             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9566         \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
9567         \dim_set:Nn \l_tmpa_dim
9568         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9569         \str_case:nn { #1 }
9570         {
9571             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9572             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9573             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9574             }
9575         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

9576         \dim_set:Nn \l_tmpb_dim
9577         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9578         \str_case:nn { #2 }
9579         {
9580             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9581             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9582             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9583         }
9584         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9585         \pgfusepathqstroke
9586         \group_end:
9587     }
9588     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
9589 }

```

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).

```

9590     \str_if_empty:NF \l_@@_submatrix_name_str
9591     {
9592         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9593         \l_@@_x_initial_dim \l_@@_y_initial_dim
9594         \l_@@_x_final_dim \l_@@_y_final_dim
9595     }
9596     \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`.

```

9597     \begin { pgfscope }
9598     \pgftransformshift
9599     {
9600         \pgfpoint
9601         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9602         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9603     }
9604     \str_if_empty:NTF \l_@@_submatrix_name_str
9605     { \@@_node_left:nn #1 { } }
9606     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9607     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9608     \pgftransformshift
9609     {
9610         \pgfpoint
9611         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9612         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9613     }
9614     \str_if_empty:NTF \l_@@_submatrix_name_str

```

```

9615 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9616 {
9617   \@@_node_right:nnnn #2
9618   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9619 }

```

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`.

```

9620 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9621 \flag_clear_new:N \l_@@_code_flag
9622 \l_@@_code_tl
9623 }

```

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`.

```

9624 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

9625 \cs_new:Npn \@@_pgfpointanchor:n #1
9626 { \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`).

```

9627 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9628 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9629 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9630 {

```

The command `\str_if_empty:nTF` is "fully expandable".

```

9631 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9632 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9633 { \@@_pgfpointanchor_ii:n { #1 } }
9634 }

```

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`).

```

9635 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9636 { \@@_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`.

```

9637 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```



```

9638 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9639 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9640 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9641 { \@@_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`).

```

9642 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9643 }

```

The following function is for the case when the name contains an hyphen.

```

9644 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9645 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9646 \@@_env:
9647 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9648 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9649 }

```

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.

```

9650 \tl_const:Nn \c_@@_integers_alist_tl
9651 {
9652 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9653 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9654 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9655 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9656 }

```

```

9657 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9658 {

```

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`.

```

9659 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9660 {
9661 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9662 \@@_env: -
9663 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9664 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9665 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9666 }
9667 {
9668 \str_if_eq:eeTF { #1 } { last }
9669 {
9670 \flag_raise:N \l_@@_code_flag
9671 \@@_env: -
9672 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9673 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9674 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9675 }

```

```

9676         { #1 }
9677     }
9678 }

```

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`).

```

9679 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9680 {
9681   \pgfnode
9682     { rectangle }
9683     { east }
9684     {
9685       \nullfont
9686       $ % $
9687       \@@_color:o \l_@@_delimiters_color_tl
9688       \left #1
9689       \vcenter
9690         {
9691           \nullfont
9692           \hrule \@height \l_tmpa_dim
9693             \c_zero_dim
9694             \c_zero_dim
9695         }
9696       \right .
9697       $ % $
9698     }
9699     { #2 }
9700     { }
9701 }

```

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.

```

9702 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
9703 {
9704   \pgfnode
9705     { rectangle }
9706     { west }
9707     {
9708       \nullfont
9709       $ % $
9710       \colorlet { current-color } { . }
9711       \@@_color:o \l_@@_delimiters_color_tl
9712       \left .
9713       \vcenter
9714         {
9715           \nullfont
9716           \hrule \@height \l_tmpa_dim
9717             \c_zero_dim
9718             \c_zero_dim
9719         }
9720       \right #1
9721       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9722       ^ { \color { current-color } \smash { #4 } }
9723       $ % $
9724     }
9725     { #2 }
9726     { }
9727 }

```

### 33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9728 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9729 {
9730   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9731   \ignorespaces
9732 }
9733 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9734 {
9735   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9736   \ignorespaces
9737 }
9738 \keys_define:nn { nicematrix / Brace }
9739 {
9740   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9741   left-shorten .value_forbidden:n = true ,
9742   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9743   right-shorten .value_forbidden:n = true ,
9744   shorten .meta:n = { left-shorten , right-shorten } ,
9745   shorten .value_forbidden:n = true ,
9746   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9747   color .tl_set:N = \l_tmpa_tl ,
9748   color .value_required:n = true ,
9749   unknown .code:n =
9750     \@@_unknown_key:nn
9751     { nicematrix / Brace }
9752     { Unknown~key~for~Brace }
9753 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#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`.

```

9754 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9755 {
9756   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9757   \@@_compute_i_j:nn { #1 } { #2 }
9758   \bool_lazy_or:nnTF
9759     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9760     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9761   {
9762     \str_if_eq:eeTF { #5 } { under }
9763     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9764     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9765   }
9766   {
9767     \tl_clear:N \l_tmpa_tl
9768     \keys_set:nn { nicematrix / Brace } { #4 }
9769     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9770     \pgfpicture
9771     \pgfrememberpicturepositiononpagetrue
9772     \pgf@relevantforpicturesizefalse
9773     \bool_if:NT \l_@@_brace_left_shorten_bool
9774     {
9775       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9776       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9777       {
9778         \cs_if_exist:cT

```

```

9779         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9780         {
9781             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9782
9783             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9784             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9785         }
9786     }
9787 }
9788 \bool_lazy_or:nnT
9789 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9790 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9791 {
9792     \@@_qpoint:n { col - \l_@@_first_j_tl }
9793     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9794 }
9795 \bool_if:NT \l_@@_brace_right_shorten_bool
9796 {
9797     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9798     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9799     {
9800         \cs_if_exist:cT
9801         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9802         {
9803             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9804             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9805             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9806         }
9807     }
9808 }
9809 \bool_lazy_or:nnT
9810 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9811 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9812 {
9813     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9814     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9815 }
9816 \pgfset { inner~sep = \c_zero_dim }
9817 \str_if_eq:eeTF { #5 } { under }
9818 { \@@_underbrace_i:n { #3 } }
9819 { \@@_overbrace_i:n { #3 } }
9820 \endpgfpicture
9821 }
9822 \group_end:
9823 }

```

The argument is the text to put above the brace.

```

9824 \cs_new_protected:Npn \@@_overbrace_i:n #1
9825 {
9826     \@@_qpoint:n { row - \l_@@_first_i_tl }
9827     \pgftransformshift
9828     {
9829         \pgfpoint
9830         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9831         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9832     }
9833     \pgfnode
9834     { rectangle }
9835     { south }
9836     {
9837         \vtop
9838         {
9839             \group_begin:
9840             \everycr { }

```

```

9841 \halign
9842 {
9843   \hfil ## \hfil \crcr
9844   \bool_if:NTF \l_@@_tabular_bool
9845     { \begin { tabular } { c } #1 \end { tabular } }
9846     { $ \begin { array } { c } #1 \end { array } $ }
9847   \cr
9848   $ % $
9849   \overbrace
9850     {
9851       \hbox_to_wd:nn
9852         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9853         { }
9854     }
9855     $ % $
9856   \cr
9857 }
9858 \group_end:
9859 }
9860 }
9861 { }
9862 { }
9863 }

```

The argument is the text to put under the brace.

```

9864 \cs_new_protected:Npn \@@_underbrace_i:n #1
9865 {
9866   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9867   \pgftransformshift
9868     {
9869       \pgfpoint
9870         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9871         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9872     }
9873   \pgfnode
9874     { rectangle }
9875     { north }
9876     {
9877       \group_begin:
9878       \everycr { }
9879       \vbox
9880         {
9881           \halign
9882             {
9883               \hfil ## \hfil \crcr
9884               $ % $
9885               \underbrace
9886                 {
9887                   \hbox_to_wd:nn
9888                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9889                     { }
9890                 }
9891               $ % $
9892             \cr
9893             \bool_if:NTF \l_@@_tabular_bool
9894               { \begin { tabular } { c } #1 \end { tabular } }
9895               { $ \begin { array } { c } #1 \end { array } $ }
9896             \cr
9897           }
9898         }
9899       \group_end:
9900     }
9901   { }
9902   { }

```

```
9903 }
```

## 34 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```
9904 \AddToHook { package / tikz / after }
9905 {
9906   \tikzset
9907   {
9908     nicematrix / brace / .style =
9909     {
9910       decoration = { brace , raise = -0.15 em } ,
9911       decorate ,
9912     } ,
```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```
9913     nicematrix / mirrored-brace / .style =
9914     {
9915       nicematrix / brace ,
9916       decoration = mirror ,
9917     }
9918   }
9919 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```
9920 \keys_define:nn { nicematrix / Hbrace }
9921 {
9922   color .code:n = ,
9923   horizontal-label .code:n = ,
9924   horizontal-labels .code:n = ,
9925   shorten .code:n = ,
9926   shorten-start .code:n = ,
9927   shorten-end .code:n = ,
9928   shorten+ .code:n = ,
9929   shorten-start+ .code:n = ,
9930   shorten-end+ .code:n = ,
9931   shorten~+ .code:n = ,
9932   shorten-start~+ .code:n = ,
9933   shorten-end~+ .code:n = ,
9934   brace-shift .code:n = ,
9935   brace-shift+ .code:n = ,
9936   brace-shift~+ .code:n = ,
9937   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9938 }
```

Here we need an “fully expandable” command.

```
9939 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9940 {
9941   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9942   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9943   { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9944 }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9945 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9946 {
9947   \int_compare:nNnTF \c@iRow < { 2 }
9948   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9949   \str_if_eq:nnTF { #2 } { * }
9950   {
9951     \bool_set_true:N \l_@@_nullify_dots_bool
9952     \Ldots
9953     [
9954       line-style = nicematrix / brace ,
9955       #1 ,
9956       up =
9957       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9958     ]
9959   }
9960   {
9961     \Hdotsfor
9962     [
9963       line-style = nicematrix / brace ,
9964       #1 ,
9965       up =
9966       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9967     ]
9968     { #2 }
9969   }
9970 }
9971 {
9972   \str_if_eq:nnTF { #2 } { * }
9973   {
9974     \bool_set_true:N \l_@@_nullify_dots_bool
9975     \Ldots
9976     [
9977       line-style = nicematrix / mirrored-brace ,
9978       #1 ,
9979       down =
9980       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9981     ]
9982   }
9983   {
9984     \Hdotsfor
9985     [
9986       line-style = nicematrix / mirrored-brace ,
9987       #1 ,
9988       down =
9989       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9990     ]
9991     { #2 }
9992   }
9993 }
9994 \keys_set:nn { nicematrix / Hbrace } { #1 }
9995 }

```

```

9996 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9997 {
9998   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9999   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10000   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
10001 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not.

```

10002 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
10003 {
10004   \int_compare:nNnTF \c@jCol < { 2 }
10005   {
10006     \str_if_eq:nnTF { #2 } { * }
10007     {
10008       \bool_set_true:N \l_@@_nullify_dots_bool
10009       \Vdots
10010       [
10011         Vbrace ,
10012         line-style = nicematrix / mirrored-brace ,
10013         #1 ,
10014         down =
10015         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10016       ]
10017     }
10018     {
10019       \Vdotsfor
10020       [
10021         Vbrace ,
10022         line-style = nicematrix / mirrored-brace ,
10023         #1 ,
10024         down =
10025         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10026       ]
10027       { #2 }
10028     }
10029   }
10030   {
10031     \str_if_eq:nnTF { #2 } { * }
10032     {
10033       \bool_set_true:N \l_@@_nullify_dots_bool
10034       \Vdots
10035       [
10036         Vbrace ,
10037         line-style = nicematrix / brace ,
10038         #1 ,
10039         up =
10040         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10041       ]
10042     }
10043     {
10044       \Vdotsfor
10045       [
10046         Vbrace ,
10047         line-style = nicematrix / brace ,
10048         #1 ,
10049         up =
10050         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10051       ]
10052       { #2 }
10053     }
10054   }
10055   \keys_set:nn { nicematrix / Hbrace } { #1 }
10056 }

```



## 35 The command TikzEveryCell

```

10057 \bool_new:N \l_@@_not_empty_bool
10058 \bool_new:N \l_@@_empty_bool
10059
10060 \keys_define:nn { nicematrix / TikzEveryCell }
10061 {
10062   not-empty .code:n =
10063     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10064     { \bool_set_true:N \l_@@_not_empty_bool }
10065     { \@@_error:n { detection-of-empty-cells } } ,
10066   not-empty .value_forbidden:n = true ,
10067   empty .code:n =
10068     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10069     { \bool_set_true:N \l_@@_empty_bool }
10070     { \@@_error:n { detection-of-empty-cells } } ,
10071   empty .value_forbidden:n = true ,
10072   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10073 }
10074
10075
10076 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10077 {
10078   \IfPackageLoadedTF { tikz }
10079   {
10080     \group_begin:
10081     \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.

```

10082     \tl_set:Nn \l_tmpa_tl { { #2 } }
10083     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10084     { \@@_for_a_block:nnnnn ##1 }
10085     \@@_all_the_cells:
10086     \group_end:
10087   }
10088   { \@@_error:n { TikzEveryCell-without-tikz } }
10089 }
10090
10091
10092 \cs_new_protected:Nn \@@_all_the_cells:
10093 {
10094   \int_step_inline:nn \c@iRow
10095   {
10096     \int_step_inline:nn \c@jCol
10097     {
10098       \cs_if_exist:cF { cell - ##1 - #####1 }
10099       {
10100         \clist_if_in:NcF \l_@@_corners_cells_clist
10101         { ##1 - #####1 }
10102         {
10103           \bool_set_false:N \l_tmpa_bool
10104           \cs_if_exist:cTF
10105           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10106           {
10107             \bool_if:NF \l_@@_empty_bool
10108             { \bool_set_true:N \l_tmpa_bool }
10109           }
10110           {
10111             \bool_if:NF \l_@@_not_empty_bool
10112             { \bool_set_true:N \l_tmpa_bool }
10113           }
10114           \bool_if:NT \l_tmpa_bool

```

```

10115         {
10116             \@@_block_tikz:nnnnn
10117             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10118         }
10119     }
10120 }
10121 }
10122 }
10123 }
10124
10125 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10126 {
10127     \bool_if:NF \l_@@_empty_bool
10128     {
10129         \@@_block_tikz:nnnnn
10130         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10131     }
10132     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10133 }
10134
10135 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10136 {
10137     \int_step_inline:nnn { #1 } { #3 }
10138     {
10139         \int_step_inline:nnn { #2 } { #4 }
10140         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10141     }
10142 }

```

## 36 The key draw-trees-in-col

```

10143 \cs_new_protected:Npn \@@_draw_trees:
10144 {
10145     \bool_if:NTF \l_@@_no_cell_nodes_bool
10146     { \@@_error:n { draw-trees-with-no-cell-nodes } }
10147     \@@_draw_trees_i:
10148 }
10149 \cs_new_protected:Npn \@@_draw_trees_i:
10150 {
10151     \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10152     \dim_zero_new:N \l_@@_em_dim
10153     \dim_set:Nn \l_@@_em_dim { 1 em }
10154     \dim_zero_new:N \l_@@_ex_dim
10155     \dim_set:Nn \l_@@_ex_dim { 1 ex }
10156     \pgfpicture
10157     \pgfrememberpicturepositiononpagetrue
10158     \pgf@relevantforpicturesizefalse
10159     \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10160     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10161     \@@_color:o \l_@@_trees_color_tl
10162     \pgfsetcornersarced
10163     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10164     \clist_map_function:NN \g_@@_col_with_trees_clist
10165     \@@_draw_trees_in_col:n
10166     \clist_gclear:N \g_@@_col_with_trees_clist
10167     \endpgfpicture
10168 }
10169 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10170 {

```

The argument is provided by curryfication.

```

10171 \int_compare:nNnTF { #1 } > \c@jCol
10172 { \@@_error:nn { Col~outside~tabular~in~trees } }
10173 {
10174   \int_compare:nNnTF { #1 } = \c@jCol
10175   { \@@_error:nn { Last~col~in~trees } }
10176   \@@_draw_trees_in_col_i:n
10177 }
10178 { #1 }
10179 }

```

```

10180 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10181 {
10182   \int_set:Nn \l_tmpa_int { 1 }
10183   \int_step_inline:nn { \c@iRow + 1 }
10184   {
10185     \cs_if_exist:cT
10186     { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10187     {
10188       \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10189       {

```

Now, you will, potentially, draw a tree.

```

10190       \@@_draw_tree:nee
10191       { #1 }
10192       { \int_use:N \l_tmpa_int }
10193       { \int_eval:n { ##1 - 1 } }
10194     }
10195     \int_set:Nn \l_tmpa_int { ##1 }
10196   }
10197 }
10198 \int_compare:nNnT \c@iRow > \l_tmpa_int
10199 {
10200   \@@_draw_tree:nee
10201   { #1 }
10202   { \int_use:N \l_tmpa_int }
10203   { \int_eval:n { \c@iRow } }
10204 }
10205 }

```

#1 is the number of column; #2 is the first row : the root of the tree; #3 is the last row of the blank zone where we will draw our tree.

```

10206 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10207 {

```

\l\_tmpa\_dim will be the  $x$ -value of the vertical rule that we will draw.

```

10208   \pgfpointanchor { \@@_env: - #1 } { 5 }
10209   \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

We will begin by the *last* branch of the tree. When that last branch has been drawn (with the vertical line), we will rise the boolean \l\_tmpa\_bool.

```

10210   \bool_set_false:N \l_tmpa_bool
10211   \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10212   {
10213     \cs_if_exist:cT
10214     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10215     {
10216       \pgfpointanchor
10217       { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10218       { base~west }
10219       \pgfpathmoveto
10220       {
10221         \pgfpoint
10222         { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }

```

```

10223         { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10224     }
10225     \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10226     \bool_if:NF \l_tmpa_bool
10227     {
10228         \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10229         \pgfpathlineto
10230             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10231         \bool_set_true:N \l_tmpa_bool
10232     }
10233     \pgfusepath { stroke }
10234 }
10235 }
10236 }
10237 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

## 37 The key create-blocks-in-col

```

10238 \cs_new_protected:Npn \@@_create_blocks_in_col:
10239 {
10240     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10241     \clist_map_inline:Nn \g_@@_cbic_clist
10242     {
10243         \cs_set:cpn
10244         {
10245             pgf @ sh @ ns @ \@@_env:
10246             - \int_eval:n { \c@iRow + 1 } - ##1 }
10247         { rien }
10248
10249         \l_tmpa_int will be the first row of the block being created.
10250         \int_set:Nn \l_tmpa_int { 1 }
10251         \int_step_inline:nn { \c@iRow + 1 }
10252         {
10253             \cs_if_exist:cT
10254             { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10255             {
10256                 \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10257                 {
10258                     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10259                     {
10260                         { \int_use:N \l_tmpa_int }
10261                         { ##1 }
10262                         { \int_eval:n { #####1 - 1 } }
10263                         { ##1 }
10264                         { }
10265                     }
10266                 }
10267             }
10268             \int_set:Nn \l_tmpa_int { #####1 }
10269         }
10270     }
10271     \clist_gclear:N \g_@@_cbic_clist
10272 }

```

## 38 The command \ShowCellNames

When the command `\ShowCellNames` is used in the `\CodeBefore`, we want to stroke the names of the cells *after* the potential backgrounds of the cells. That's why we add the command `\@@_ShowCellNames` to the right of the command `\@@_actually_color:` which actually fill the backgrounds.

```

10271 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10272 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10273 \NewDocumentCommand \@@_ShowCellNames { }
10274 {
10275   \bool_if:NT \l_@@_in_code_after_bool
10276   {
10277     \pgfpicture
10278     \pgfrememberpicturepositiononpagetrue
10279     \pgf@relevantforpicturesizefalse
10280     \pgfpathrectanglecorners
10281     { \@@_qpoint:n { 1 } }
10282     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10283     \pgfsetfillopacity { 0.75 }
10284     \pgfsetfillcolor { white }
10285     \pgfusepathqfill
10286     \endpgfpicture
10287   }
10288   \dim_gzero_new:N \g_@@_tmpc_dim
10289   \dim_gzero_new:N \g_@@_tmpd_dim
10290   \dim_gzero_new:N \g_@@_tmpe_dim
10291   \int_step_inline:nn \c@iRow
10292   {
10293     \bool_if:NTF \l_@@_in_code_after_bool
10294     {
10295       \pgfpicture
10296       \pgfrememberpicturepositiononpagetrue
10297       \pgf@relevantforpicturesizefalse
10298     }
10299     { \begin { pgfpicture } }
10300     \@@_qpoint:n { row - ##1 }
10301     \dim_set_eq:NN \l_tmpa_dim \pgf@y
10302     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10303     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10304     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10305     \bool_if:NTF \l_@@_in_code_after_bool
10306     { \endpgfpicture }
10307     { \end { pgfpicture } }
10308     \int_step_inline:nn \c@jCol
10309     {
10310       \hbox_set:Nn \l_tmpa_box
10311       {
10312         \normalfont \Large \sffamily \bfseries
10313         \bool_if:NTF \l_@@_in_code_after_bool
10314         { \color { red } }
10315         { \color { red ! 50 } }
10316         ##1 - ####1
10317       }
10318       \bool_if:NTF \l_@@_in_code_after_bool
10319       {
10320         \pgfpicture
10321         \pgfrememberpicturepositiononpagetrue
10322         \pgf@relevantforpicturesizefalse
10323       }
10324       { \begin { pgfpicture } }
10325       \@@_qpoint:n { col - ####1 }
10326       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10327       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10328       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10329       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10330       \bool_if:NTF \l_@@_in_code_after_bool
10331       { \endpgfpicture }
10332       { \end { pgfpicture } }

```

```

10333 \fp_set:Nn \l_tmpa_fp
10334 {
10335     \fp_min:nn
10336     {
10337         \fp_min:nn
10338         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10339         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10340     }
10341     { 1.0 }
10342 }
10343 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10344 \pgfpicture
10345 \pgfrememberpicturepositiononpagetrue
10346 \pgf@relevantforpicturesizefalse
10347 \pgftransformshift
10348 {
10349     \pgfpoint
10350     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10351     \g_tmpa_dim
10352 }
10353 \pgfnode
10354 { rectangle }
10355 { center }
10356 { \box_use:N \l_tmpa_box }
10357 { }
10358 { }
10359 \endpgfpicture
10360 }
10361 }
10362 }

```

## 39 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

10363 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to true if the option `footnotehyper` is used.

```

10364 \bool_new:N \g_@@_footnote_bool
10365 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10366 {
10367     You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
10368     but~that~key~is~unknown. \\
10369     It~will~be~ignored. \\
10370     For~a~list~of~the~available~keys,~type~H~<return>.
10371 }
10372 {
10373     The~available~keys~are~(in~alphabetic~order):~
10374     footnote,~
10375     footnotehyper,~
10376     messages~for~Overleaf,~
10377     renew~dots~and~

```

```

10378     renew-matrix.
10379 }
10380 \keys_define:nn { nicematrix }
10381 {
10382     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10383     renew-dots .value_forbidden:n = true ,
10384     renew-matrix .code:n = \@@_renew_matrix: ,
10385     renew-matrix .value_forbidden:n = true ,
10386     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10387     footnote .bool_set:N = \g_@@_footnote_bool ,
10388     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10389     unknown .code:n = \@@_error:n { Unknown~key~for~package }
10390 }
10391 \ProcessKeyOptions

10392 \@@_msg_new:nn { footnote-with-footnotehyper-package }
10393 {
10394     You~can't~use~the~option~'footnote'~because~the~package~
10395     footnotehyper~has~already~been~loaded.~
10396     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10397     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10398     of~the~package~footnotehyper.\\
10399     The~package~footnote~won't~be~loaded.
10400 }
10401 \@@_msg_new:nn { footnotehyper-with-footnote-package }
10402 {
10403     You~can't~use~the~option~'footnotehyper'~because~the~package~
10404     footnote~has~already~been~loaded.~
10405     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10406     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10407     of~the~package~footnote.\\
10408     The~package~footnotehyper~won't~be~loaded.
10409 }

10410 \bool_if:NT \g_@@_footnote_bool
10411 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10412     \IfClassLoadedTF { beamer }
10413     { \bool_set_false:N \g_@@_footnote_bool }
10414     {
10415         \IfPackageLoadedTF { footnotehyper }
10416         { \@@_error:n { footnote-with-footnotehyper~package } }
10417         { \usepackage { footnote } }
10418     }
10419 }

10420 \bool_if:NT \g_@@_footnotehyper_bool
10421 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10422     \IfClassLoadedTF { beamer }
10423     { \bool_set_false:N \g_@@_footnote_bool }
10424     {
10425         \IfPackageLoadedTF { footnote }
10426         { \@@_error:n { footnotehyper-with-footnote~package } }
10427         { \usepackage { footnotehyper } }
10428     }
10429     \bool_set_true:N \g_@@_footnote_bool
10430 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

## 40 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

10431 \bool_new:N \l_@@_underscore_loaded_bool
10432 \IfPackageLoadedT { underscore }
10433 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10434 \AtBeginDocument
10435 {
10436   \bool_if:NF \l_@@_underscore_loaded_bool
10437   {
10438     \IfPackageLoadedT { underscore }
10439     { \@@_error:n { underscore~after~nicematrix } }
10440   }
10441 }

```

## 41 Compatibility with threeparttable

```

10442 \AtBeginDocument
10443 {
10444   \IfPackageLoadedT { threeparttable }
10445   {
10446     \AddToHook { env / threeparttable / begin }
10447     {
10448       \TPT@hookin { NiceTabular }
10449       \TPT@hookin { NiceTabular* }
10450       \TPT@hookin { NiceTabularX }
10451     }
10452   }
10453 }

```

## 42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is `+`.

`#1` is a clist of names of sets of keys and `#2` is the error message to send.

```

10454 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10455 {
10456   \str_if_eq:eeTF
10457   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10458   { + }
10459   {
10460     \str_set:Ne \l_tmpa_str
10461     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10462     \bool_set_false:N \l_tmpa_bool
10463     \clist_map_inline:nn { #1 }
10464     {

```



```

10465         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10466         {
10467             \@@_error:n { key~without~+~exists }
10468             \bool_set_true:N \l_tmpa_bool
10469             \clist_map_break:
10470         }
10471     }
10472     \bool_if:NF \l_tmpa_bool
10473     {
10474         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10475         \@@_unknown_key_i:nn { #1 } { #2 }
10476     }
10477 }
10478 { \@@_unknown_key_i:nn { #1 } { #2 } }
10479 }

```

We try a normalisation of the name of the key, and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of nicematrix).

**#1** is a clist of names of sets of keys and **#2** is the error message to send.

```

10480 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10481 {
10482     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10483     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10484     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10485     \bool_set_false:N \l_tmpa_bool
10486     \clist_map_inline:nn { #1 }
10487     {
10488         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10489         {
10490             \@@_error:n { key~with~normal~form~exists }
10491             \bool_set_true:N \l_tmpa_bool
10492             \clist_map_break:
10493         }
10494     }
10495     \bool_if:NF \l_tmpa_bool
10496     {
10497         \@@_error:n { #2 }

```

If `messages-for-Overleaf` is not in force, the list of the available keys is not written in the main error message but only in the complement (if the final user presses H). That's why we write, in all circumstances, the list of the available keys in order to facilitate the work of the systems which analyze the error by IA (such as Prism).

```

10498         \bool_if:NF \g_@@_messages_for_Overleaf_bool
10499         { \msg_info:nn { nicematrix } { #2~+ } }
10500     }
10501 }
10502 \@@_msg_new:nn { key~without~+~exists }
10503 {
10504     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10505     additive~syntax~(with~+=)~.~It~will~be~ignored.
10506 }
10507 \@@_msg_new:nn { key~with~normal~form~exists }
10508 {
10509     No~key~'\l_keys_key_str'.\\
10510     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10511 }
10512 \str_const:Ne \c_@@_available_keys_str
10513 {
10514     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }

```

```

10515     { For-a-list-of-the-available-keys,~type-H~<return>. }
10516 }
10517 \seq_new:N \g_@@_types_of_matrix_seq
10518 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10519 {
10520     NiceMatrix ,
10521     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10522 }
10523 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10524 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

10525 \cs_new_protected:Npn \@@_err_too_many_cols:
10526 {
10527     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10528     { \@@_fatal:nn { too-many-cols-for-array } }
10529     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10530     { \@@_fatal:n { too-many-cols-for-matrix } }
10531     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10532     { \@@_fatal:n { too-many-cols-for-matrix } }
10533     \bool_if:NF \l_@@_last_col_without_value_bool
10534     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10535 }

```

The following command must *not* be protected since it's used in an error message.

```

10536 \cs_new:Npn \@@_message_hdotsfor:
10537 {
10538     \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
10539     { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
10540       \token_to_str:N \Hbrace \ is-incorrect. }
10541 }
10542 \cs_new_protected:Npn \@@_Hline_in_cell:
10543 { \@@_error:n { Misuse-of-Hline } }
10544 \@@_msg_new:nn { Misuse-of-Hline }
10545 {
10546     Misuse-of~\token_to_str:N \Hline. \\
10547     Error-in-the-row~ \int_use:N \c@iRow\ of-your~\@@_full_name_env:~
10548     \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must-be-used-only~
10549     at-the-beginning-of-a-row.~Your-command-will-be-ignored.
10550 }
10551 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10552 {
10553     Incompatible-options.\\
10554     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-the-same-time.~
10555     The-output-will-not-be-reliable.
10556 }
10557 \@@_msg_new:nn { Body-alone }
10558 {
10559     \token_to_str:N \Body\ alone. \\
10560     You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10561 }
10562 \@@_msg_new:nn { Bad-use-of~CodeBefore }
10563 {
10564     Bad-use-of~\token_to_str:N \CodeBefore. \\
10565     \token_to_str:N \CodeBefore\ must-be-used-only-at-the-beginning-of~
10566     the-environment.
10567 }

```

```

10568 \@@_msg_new:nn { NiceTabularX-probably-required }
10569 {
10570   Incorrect-syntax.\
10571   You-probably-want-to-use-\{NiceTabularX\}-whose-first-argument-is-the~
10572   ~width-that-you-wish-for-your-tabular.~But-you-can-also-use-\{NiceTabular\}~
10573   with-the-key-'width'.
10574 }
10575 \@@_msg_new:nn { cellcolor-in-Block }
10576 {
10577   Bad-use-of-\token_to_str:N \cellcolor \
10578   You-can't-use-\token_to_str:N \cellcolor\ in-\token_to_str:N \Block\
10579   \bool_if:NTF \l_@@_amp_in_blocks_bool
10580   { (but-you-could-use-it-in-a-sub-block-since-'&-in-blocks'-is-in-force) }
10581   { (it's-possible-in-a-sub-block-when-'&-in-blocks'-is-in-force) }
10582   .~Here,~you-should-use-the-key-'fill'-of-the-block.~Your-command-will-be-ignored.
10583 }
10584 \@@_msg_new:nn { rowcolor-in-Block }
10585 {
10586   Bad-use-of-\token_to_str:N \rowcolor \
10587   You-can't-use-\token_to_str:N \rowcolor\ in-\token_to_str:N \Block.~
10588   However,~it's-possible-to-color-the-block-with-its-key-'fill'.~
10589   Your-command-will-be-ignored.
10590 }
10591 \@@_msg_new:nn { key-color-inside }
10592 {
10593   Deleted-key.\
10594   The-key-'color-inside'~(and-its-alias-'colortbl-like')~has-been-deleted-in
10595   ~'nicematrix'~and-must-not-be-used.
10596 }
10597 \@@_msg_new:nn { Invalid-argument-for-w }
10598 {
10599   Invalid-argument-for-w.\
10600   You-have-used-the-type-of-alignment~'#1'~for-your-column~'w'~
10601   but-only~'c'~,~'r'~,~'l'~and~'s'~are-allowed-in-a-column~'w'.~
10602   If-you-go-on,~'c'~will-be-used.
10603 }
10604 \@@_msg_new:nn { invalid-weight }
10605 {
10606   Unknown-key.\
10607   The-key~'\l_keys_key_str'~of-your-column-X-is-unknown-and-will-be-ignored.~
10608   The-available-keys-are:~l,~c,~r,~t(=p),~m,~b,~V~
10609   \IfPackageLoadedTF { varwidth }
10610   { (since-'varwidth'-is-loaded)~}
10611   { (if-you-load-'varwidth')~}
10612   and-real-numbers-for-the-weight-of-the-X-column.
10613 }
10614 \@@_msg_new:nn { last-col-not-used }
10615 {
10616   Column-not-used.\
10617   The-key-'last-col'~is-in-force-but-you-have-not-used-that-last-column~
10618   in-your~\@@_full_name_env: .~However,~you-can-go-on.
10619 }
10620 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10621 {
10622   Too-many-columns.\
10623   In-the-row~\int_eval:n { \c@iRow },~
10624   you-try-to-use-more-columns~
10625   than-allowed-by-your~\@@_full_name_env: .
10626   \@@_message_hdotsfor: \
10627   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
10628   (plus-the-exterior-columns).~

```

```

10629 But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10630 }
10631 \@@_msg_new:nn { too-many-cols-for-matrix }
10632 {
10633   Too-many-columns.\.
10634   In-the-row~ \int_eval:n { \c@iRow } ,~
10635   you-try-to-use-more-columns-than-allowed-by-your~ \@@_full_name_env: .
10636   \@@_message_hdotsfor: \
10637   Recall-that-the-maximal-number-of-columns-for-a-matrix~
10638   (excepted-the-potential-exterior-columns)-is-fixed-by-the~
10639   LaTeX-counter-'MaxMatrixCols'.~
10640   Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
10641   (use~ \token_to_str:N \setcounter \ to-change-that-value).~
10642   But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10643 }
10644 \@@_msg_new:nn { too-many-cols-for-array }
10645 {
10646   Too-many-columns.\.
10647   In-the-row~ \int_eval:n { \c@iRow } ,~
10648   ~you-try-to-use-more-columns-than-allowed-by-your~
10649   \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
10650   \int_use:N \g_@@_static_num_of_col_int \
10651   \bool_if:nT
10652     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10653     { (plus-the-exterior-ones)-}
10654   since-the-preamble-is~' \g_@@_user_preamble_tl '.~
10655   But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10656 }
10657 \@@_msg_new:nn { columns-not-used }
10658 {
10659   Columns-not-used.\.
10660   The-preamble-of-your~ \@@_full_name_env: \ is~
10661   '\exp_not:V \g_@@_user_preamble_tl'.~
10662   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
10663   columns-but-you-only-used~ \int_use:N \c@jCol .~The-columns-you-did-not~
10664   used-won't-be-created.~You-won't-have-similar-warning-till-the-end-of-the-document.
10665 }
10666 }
10667 \@@_msg_new:nn { Bad-use-of-NiceTabularNotes }
10668 {
10669   Bad-use-of-\token_to_str:N \NiceTabularNotes \.
10670   \token_to_str:N\NiceTabularNotes\ should-be-used-only-after-at-tabular~
10671   which-uses~`notes/no-print`.~ Your-command-will-be-ignored.
10672 }
10673 \@@_msg_new:nn { empty-preamble }
10674 {
10675   Empty-preamble.\.
10676   The-preamble-of-your~ \@@_full_name_env: \ is-empty.
10677 }
10678 \@@_msg_new:nn { in-first-col }
10679 {
10680   Erroneous-use.\.
10681   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.~
10682   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-col'.
10683 }
10684 \@@_msg_new:nn { in-last-col }
10685 {
10686   Erroneous-use.\.
10687   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.~
10688   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-col'.
10689 }

```

```

10690 \@@_msg_new:nn { in~first~row }
10691 {
10692   Erroneous~use.\\
10693   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
10694   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first~row'.
10695 }
10696 \@@_msg_new:nn { in~last~row }
10697 {
10698   Erroneous~use.\\
10699   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
10700   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last~row'.
10701 }
10702 \@@_msg_new:nn { TopRule~without~booktabs }
10703 {
10704   Erroneous~use.\\
10705   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
10706   You~should~load~'booktabs'~(before~or~after~'nicematrix').~
10707   Your~command~will~be~ignored.
10708 }
10709 \@@_msg_new:nn{ rotate~in~p~col }
10710 {
10711   \token_to_str:N \rotate\ forbidden.\\
10712   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10713   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~
10714   If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
10715   consider~the~classical~command~\token_to_str:N \rotatebox.
10716 }
10717 \@@_msg_new:nn { TopRule~without~tikz }
10718 {
10719   Erroneous~use.\\
10720   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.~
10721   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
10722   \IfPackageLoadedF { booktabs }
10723     { You~should~also~load~'booktabs'~
10724       with~\token_to_str:N \usepackage \{booktabs\}.~ }
10725   Your~command~will~be~ignored.
10726 }
10727 \@@_msg_new:nn { caption~outside~float }
10728 {
10729   Key~caption~forbidden.\\
10730   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10731   environment~(such~as~\{table\}).~Your~key~will~be~ignored.
10732 }
10733 \@@_msg_new:nn { short~caption~without~caption }
10734 {
10735   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10736   However,~your~'short~caption'~will~be~used~as~'caption'.
10737 }
10738 \@@_msg_new:nn { double~closing~delimiter }
10739 {
10740   Double~delimiter.\\
10741   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10742   delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
10743   \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10744 }
10745 \@@_msg_new:nn { delimiter~after~opening }
10746 {
10747   Double~delimiter.\\
10748   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10749   delimiter.~That~delimiter~will~be~ignored.
10750 }

```

```

10751 \@@_msg_new:nn { bad-option-for-line-style }
10752 {
10753   Bad-line-style.\
10754   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line-style'~
10755   is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
10756   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
10757 }
10758 \@@_msg_new:nn { draw-trees-with-no-cell-nodes }
10759 {
10760   Incompatible-keys.\
10761   You~can't~use~the~key~'draw-trees-in-col'~here~because~the~key~'no-cell-nodes'~
10762   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10763   is~to~speed~compilation~up).~If~you~go~on,~that~key~will~be~ignored.
10764 }
10765 \@@_msg_new:nn { corners-with-no-cell-nodes }
10766 {
10767   Incompatible-keys.\
10768   You~can't~use~the~key~'corners'~here~because~the~key~'no-cell-nodes'~
10769   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10770   is~to~speed~compilation~up).\
10771   If~you~go~on,~that~key~will~be~ignored.
10772 }
10773 \@@_msg_new:nn { extra-nodes-with-no-cell-nodes }
10774 {
10775   Incompatible-keys.\
10776   You~can't~create~'extra-nodes'~here~because~the~key~'no-cell-nodes'~
10777   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10778   is~to~speed~compilation~up).~If~you~go~on,~those~extra~nodes~won't~be~created.
10779 }
10780 \@@_msg_new:nn { Identical-notes-in-caption }
10781 {
10782   Identical~tabular~notes.\
10783   You~can't~put~several~notes~with~the~same~content~in~
10784   \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
10785   If~you~go~on,~the~output~will~probably~be~erroneous.
10786 }
10787 \@@_msg_new:nn { tabularnote~below~the~tabular }
10788 {
10789   \token_to_str:N \tabularnote \ forbidden\
10790   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10791   of~your~tabular~because~the~caption~will~be~composed~below~
10792   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10793   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
10794   Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
10795   no~similar~error~will~raised~in~this~document.
10796 }
10797 \@@_msg_new:nn { Unknown~key~for~rules }
10798 {
10799   Unknown~key.\
10800   There~are~only~three~keys~available~here:~'width',~'color'~and~
10801   'fix~vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10802 }
10803 \@@_msg_new:nn { Unknown~key~for~trees }
10804 {
10805   Unknown~key.\
10806   There~are~only~three~keys~available~here:~width~color~and~
10807   rounded~corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
10808 }
10809 % \end{macrocode}
10810 %
10811 %

```

```

10812 % \begin{macrocode}
10813 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10814 {
10815     Unknown~key.\\
10816     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10817     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10818     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10819     'horizontal-label(s)',~'shorten'~'shorten-end'~
10820     and~'shorten-start'.
10821 }
10822 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10823 {
10824     Unknown~key.\\
10825     There~is~only~two~keys~available~here:~
10826     'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10827 }
10828 \@@_msg_new:nn { Unknown~key~for~rotate }
10829 {
10830     Unknown~key.\\
10831     The~only~keys~available~here~are~'c'~and~'-90'.~
10832     Your~key~' \l_keys_key_str '~will~be~ignored.
10833 }
10834 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10835 {
10836     Unknown~key.\\
10837     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10838     It~you~go~on,~you~will~probably~have~other~errors. \\
10839     \c_@@_available_keys_str
10840 }
10841 {
10842     The~available~keys~are~(in~alphabetic~order):~
10843     ccommand,~
10844     color,~
10845     command,~
10846     dotted,~
10847     letter,~
10848     multiplicity,~
10849     sep~color,~
10850     tikz,~and~total~width.
10851 }
10852 \@@_msg_new:nnn { Unknown~key~for~default~line }
10853 {
10854     Unknown~key.\\
10855     The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10856     It~you~go~on,~you~will~probably~have~other~errors. \\
10857     \c_@@_available_keys_str
10858 }
10859 {
10860     The~available~keys~are~(in~alphabetic~order):~
10861     color,~
10862     dotted,~
10863     multiplicity,~
10864     sep~color,~
10865     tikz,~and~total~width.
10866 }
10867 \@@_msg_new:nnn { Unknown~key~for~xdots }
10868 {
10869     Unknown~key.\\
10870     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10871     \c_@@_available_keys_str
10872 }
10873 {

```

```

10874 The~available~keys~are~(in~alphabetic~order):~
10875 'color',~
10876 'horizontal(s)-labels',~
10877 'inter',~
10878 'line-style',~
10879 'nullify',~
10880 'radius',~
10881 'shorten',~
10882 'shorten-end'~and~'shorten-start'.
10883 }

10884 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10885 {
10886   Unknown~key.\\
10887   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10888   (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
10889 }

10890 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10891 {
10892   Error~with~'draw-trees-in-col' \\
10893   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10894   is~\int_use:N \c@jCol.~It~will~be~ignored.
10895 }

10896 \@@_msg_new:nn { Last~col~in~trees }
10897 {
10898   Error~with~'draw-trees-in-col' \\
10899   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
10900   the~last~column~of~your~tabular.~It~will~be~ignored.
10901 }

10902 \@@_msg_new:nn { label~without~caption }
10903 {
10904   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10905   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10906 }

10907 \@@_msg_new:nn { W~warning }
10908 {
10909   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10910   (row~ \int_use:N \c@iRow ).
10911 }

10912 \@@_msg_new:nn { Construct~too~large }
10913 {
10914   Construct~too~large.\\
10915   Your~command~ \token_to_str:N #1
10916   can't~be~drawn~because~your~matrix~is~too~small.~
10917   Your~command~will~be~ignored.
10918 }

10919 \@@_msg_new:nn { underscore~after~nicematrix }
10920 {
10921   Problem~with~'underscore'.\\
10922   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10923   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10924   ' \token_to_str:N \Cdots \token_to_str:N _
10925   \{ n \token_to_str:N \text \{ ~times \} \}' .
10926 }

10927 \@@_msg_new:nn { ampersand~in~light-syntax }
10928 {
10929   Ampersand~forbidden.\\
10930   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10931   ~the~key~'light-syntax'~is~in~force.
10932 }

10933 \@@_msg_new:nn { double~backslash~in~light-syntax }

```



```

10934 {
10935   Double~backslash~forbidden.\\
10936   You~can't~use~ \token_to_str:N \\
10937   ~to~separate~rows~because~the~key~'light-syntax'~
10938   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10939   (set~by~the~key~'end-of-row').
10940 }
10941 \@@_msg_new:nn { bad~value~for~baseline }
10942 {
10943   Bad~value~for~baseline.\\
10944   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10945   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10946   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10947   the~form~'line-i'.~A~value~of~1~will~be~used.
10948 }
10949 \@@_msg_new:nn { bad~value~for~baseline~line }
10950 {
10951   Bad~value~for~baseline~with~line.\\
10952   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10953   valid.~The~number~of~the~line~must~be~between~1~and~
10954   \int_eval:n { \c@iRow + 1 }.~
10955   A~value~of~'line-1'~will~be~used.
10956 }
10957 \@@_msg_new:nn { detection~of~empty~cells }
10958 {
10959   Problem~with~'not~empty'\\
10960   For~technical~reasons,~you~must~activate~
10961   'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
10962   in~order~to~use~the~key~' \l_keys_key_str '~
10963   Your~key~will~be~ignored.
10964 }
10965 \@@_msg_new:nn { siunitx~too~old }
10966 {
10967   siunitx~too~old\\
10968   You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
10969   is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26).
10970 }
10971 \@@_msg_new:nn { Invalid~name }
10972 {
10973   Invalid~name.\\
10974   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10975   \SubMatrix \ of~your~ \@@_full_name_env: .~
10976   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
10977   Your~key~will~be~ignored.
10978 }
10979 \@@_msg_new:nn { Hbrace~not~allowed }
10980 {
10981   Command~not~allowed.\\
10982   You~can't~use~the~command~ \token_to_str:N #1
10983   because~you~have~not~loaded~
10984   \IfPackageLoadedTF { tikz }
10985   { the~TikZ~library~'decorations.pathreplacing'.~Use~ }
10986   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10987   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10988   Your~command~will~be~ignored.
10989 }
10990 \@@_msg_new:nn { Vbrace~not~allowed }
10991 {
10992   Command~not~allowed.\\
10993   You~can't~use~the~command~ \token_to_str:N \Vbrace \
10994   because~you~have~not~loaded~TikZ~

```

```

10995     and-the-TikZ-library~'decorations.pathreplacing'.~
10996     Use: ~\token_to_str:N \usepackage \{tikz\}~
10997     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10998     Your~command~will~be~ignored.
10999 }
11000 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
11001 {
11002     Wrong~line.\\
11003     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
11004     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11005     number~is~not~valid.~It~will~be~ignored.
11006 }
11007 \@@_msg_new:nn { Impossible~SubMatrix }
11008 {
11009     Impossible~SubMatrix.\\
11010     It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
11011     because~all~the~cells~are~empty~in~the~column~on~the~#1~
11012     side~of~that~\token_to_str:N \SubMatrix.
11013     \bool_if:NT \l_@@_submatrix_slim_bool
11014     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
11015     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11016 }
11017 \@@_msg_new:nn { Impossible~SubMatrix~no~cell~nodes }
11018 {
11019     Impossible~SubMatrix.\\
11020     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
11021     because~the~key~'no~cell~nodes'~is~in~force.~
11022     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11023 }
11024 \@@_msg_new:nnn { width~without~X~columns }
11025 {
11026     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11027     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11028     Your~key~will~be~ignored.
11029 }
11030 {
11031     This~message~is~the~message~'width~without~X~columns'~
11032     of~the~module~'nicematrix'.~
11033     The~experimented~users~can~disable~that~message~with~
11034     \token_to_str:N \msg_redirect_name:nnn .\\
11035 }
11036
11037 \@@_msg_new:nn { key~multiplicity~with~dotted }
11038 {
11039     Incompatible~keys. \\
11040     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11041     in~a~'custom~line'.~They~are~incompatible.~
11042     The~key~'multiplicity'~will~be~ignored.
11043 }
11044 \@@_msg_new:nn { empty~environment }
11045 {
11046     Empty~environment.\\
11047     Your~ \@@_full_name_env: \ is~empty.
11048 }
11049 \@@_msg_new:nn { No~letter~and~no~command }
11050 {
11051     Erroneous~use.\\
11052     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
11053     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11054     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11055 }

```

```

11056 \@@_msg_new:nn { Forbidden~letter }
11057 {
11058   Forbidden~letter.\\
11059   You~can't~use~the~letter~'#1'~for~a~customized~line.~
11060   It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11061 }
11062 \@@_msg_new:nn { Several~letters }
11063 {
11064   Wrong~name.\\
11065   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11066   have~used~'\l_@@_letter_str '~).~It~will~be~ignored.
11067 }
11068 \@@_msg_new:nn { Delimiter~with~small }
11069 {
11070   Delimiter~forbidden.\\
11071   You~can't~put~a~delimiter~in~the~preamble~of~your~
11072   \@@_full_name_env: \
11073   because~the~key~'small'~is~in~force.
11074 }
11075 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11076 {
11077   Unknown~cell.\\
11078   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11079   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11080   can't~be~executed~because~a~cell~doesn't~exist.~
11081   Your~command~ \token_to_str:N \line \ will~be~ignored.
11082 }
11083 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11084 {
11085   Duplicate~name. \\
11086   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11087   in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11088   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11089   { For~a~list~of~the~names~already~used,~type~H~<return>. }
11090 }
11091 {
11092   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11093   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11094 }
11095 \@@_msg_new:nn { r~or~l~with~preamble }
11096 {
11097   Erroneous~use. \\
11098   You~can't~use~the~key~'\l_keys_key_str '~in~your~ \@@_full_name_env: .~
11099   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11100   your~ \@@_full_name_env: .~
11101   Your~key~will~be~ignored.
11102 }
11103 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11104 {
11105   Erroneous~use. \\
11106   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11107   in~an~exterior~column~of~the~array.
11108 }
11109 \@@_msg_new:nn { bad~corner }
11110 {
11111   Bad~corner. \\
11112   '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11113   'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11114   This~specification~of~corner~will~be~ignored.
11115 }

```

```

11116 \@@_msg_new:nn { bad-border }
11117 {
11118   Bad-border. \
11119   '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11120   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11121   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11122   also~use~the~key~'tikz'
11123   \IfPackageLoadedF { tikz }
11124   { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11125   This~specification~of~border~will~be~ignored.
11126 }
11127 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11128 {
11129   TikZ~not~loaded. \
11130   You~can't~use~ \token_to_str:N \TikzEveryCell \
11131   because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11132   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11133   Your~command~will~be~ignored.
11134 }
11135 \@@_msg_new:nn { tikz~key~without~tikz }
11136 {
11137   TikZ~not~loaded. \
11138   You~can't~use~the~key~'tikz'~for~the~command~ \token_to_str:N
11139   \Block '~because~you~have~not~loaded~TikZ.~
11140   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11141   before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11142 }
11143 \@@_msg_new:nn { Bad~argument~for~Block }
11144 {
11145   Bad~argument. \
11146   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11147   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11148   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
11149   the~argument~was~empty).
11150 }
11151 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11152 {
11153   Erroneous~use. \
11154   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11155   'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11156   (the~value~' \l_keys_value_tl '~will~be~ignored).
11157 }
11158 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11159 {
11160   Erroneous~use. \
11161   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11162   'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11163   (the~value~' \l_keys_value_tl '~will~be~ignored).
11164 }
11165 \@@_msg_new:nn { Block~too~large~1 }
11166 {
11167   Block~too~large. \
11168   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
11169   too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11170 }
11171 \@@_msg_new:nn { Block~too~large~2 }
11172 {
11173   Block~too~large. \
11174   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
11175   \g_@@_static_num_of_col_int \
11176   columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~

```

```

11177     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands-
11178     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: .~
11179     This-block-and-maybe-others-will-be-ignored.
11180 }
11181 \@@_msg_new:nn { unknown~column-type }
11182 {
11183     Bad-column-type. \\\
11184     The-column-type~'#1'~in-your~ \@@_full_name_env: \ is-unknown.
11185 }
11186 \@@_msg_new:nn { unknown~column-type-multicolumn }
11187 {
11188     Bad-column-type. \\\
11189     The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
11190     ~of-your~ \@@_full_name_env: \ is-unknown.
11191 }
11192 \@@_msg_new:nn { unknown~column-type-S }
11193 {
11194     Bad-column-type. \\\
11195     The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown.~
11196     If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
11197     load-that-package-with~\token_to_str:N \usepackage \{siunitx\}.
11198 }
11199 \@@_msg_new:nn { unknown~column-type-S-multicolumn }
11200 {
11201     Bad-column-type. \\\
11202     The-column-type~'S'~in-the-command~\token_to_str:N \multicolumn \
11203     of-your~ \@@_full_name_env: \ is-unknown.~If-you-want-to-use-the-column-
11204     type~'S'~of~siunitx,~you-should-load-that-package-with~
11205     \token_to_str:N \usepackage \{siunitx\}.
11206 }
11207 \@@_msg_new:nn { tabularnote~forbidden }
11208 {
11209     Forbidden-command. \\\
11210     You-can't-use-the-command~ \token_to_str:N \tabularnote \
11211     ~here.~This-command-is-available-only-in~
11212     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in~
11213     the-argument-of-a-command~\token_to_str:N \caption \ included-
11214     in-an-environment~\{table\}.~Your-command-will-be-ignored.
11215 }
11216 \@@_msg_new:nn { borders~forbidden }
11217 {
11218     Forbidden-key.\\
11219     You-can't-use-the-key~'borders'~of~the-command~ \token_to_str:N \Block \
11220     because-the-option~'rounded-corners'~is-in-force-with-a-non-zero-value.~
11221     Your-key-will-be-ignored.
11222 }
11223 \@@_msg_new:nn { bottomrule~without~booktabs }
11224 {
11225     booktabs-not-loaded.\\
11226     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
11227     loaded~'booktabs'.~You-should-load~'booktabs',~before-or~
11228     after~'nicematrix'.~Your-key-will-be-ignored.
11229 }
11230 \@@_msg_new:nn { enumitem~not~loaded }
11231 {
11232     enumitem-not-loaded. \\\
11233     You-can't-use-the-command~ \token_to_str:N \tabularnote \
11234     ~because-you-haven't~loaded~'enumitem'.~We-should-load-it~
11235     (before-or-after~'nicematrix').~All-the-commands~
11236     \token_to_str:N \tabularnote \ will-be-ignored-in-the-document.
11237 }

```

```

11238 \@@_msg_new:nn { tikz-without-tikz }
11239 {
11240   TikZ~not~loaded. \\
11241   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11242   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11243   If~you~go~on,~your~key~will~be~ignored.
11244 }
11245 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
11246 {
11247   TikZ~not~loaded. \\
11248   You~have~used~the~key~'tikz'~in~the~definition~of~a~
11249   customized~line~(with~'custom-line')~but~TikZ~is~not~loaded.~
11250   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11251   use~that~custom~line.~You~should~load~TikZ~with~
11252   \token_to_str:N \usepackage \{tikz\}.
11253 }
11254 \@@_msg_new:nn { tikz-in-borders-without-tikz }
11255 {
11256   TikZ~not~loaded. \\
11257   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11258   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11259   Your~key~will~be~ignored.~You~should~load~TikZ~with~
11260   \token_to_str:N \usepackage \{tikz\}
11261 }
11262 \@@_msg_new:nn { color-in-custom-line-with-tikz }
11263 {
11264   Erroneous~use.\\
11265   In~a~'custom-line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11266   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11267   The~key~'color'~will~be~ignored.
11268 }
11269 \@@_msg_new:nn { Wrong~last~row }
11270 {
11271   Wrong~number.\\
11272   You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
11273   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11274   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11275   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11276   problem~by~using~'last-row'~without~value~(more~compilations~
11277   might~be~necessary).
11278 }
11279 \@@_msg_new:nn { Yet~in~env }
11280 {
11281   Nested~environments.\\
11282   Environments~of~nicematrix~can't~be~nested.~However~you~
11283   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11284   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11285   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11286   that~box~in~another~environment~of~nicematrix.
11287 }
11288 \@@_msg_new:nn { Outside~math~mode }
11289 {
11290   Outside~math~mode.\\
11291   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11292   (and~not~in~ \token_to_str:N \vcenter ).
11293 }
11294 \@@_msg_new:nn { One~letter~allowed }
11295 {
11296   Bad~name.\\
11297   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11298   you~have~used~' \l_keys_value_tl '~Your~key~will~be~ignored.

```

```

11299 }
11300 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11301 {
11302   Environment~\{TabularNote\}~forbidden.\\
11303   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11304   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11305   \{TabularNote\}~will~be~ignored.
11306 }
11307 \@@_msg_new:nn { varwidth~not~loaded }
11308 {
11309   varwidth~not~loaded.\\
11310   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11311   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11312   Your~column~will~behave~like~'p'.
11313 }
11314 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11315 {
11316   varwidth~not~loaded.\\
11317   You~can't~use~the~key~'V'~in~your~column~'X'~
11318   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11319   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11320 }
11321 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11322 {
11323   Unknown~key.\\
11324   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11325   \c_@@_available_keys_str
11326 }
11327 {
11328   The~available~keys~are:~color,~multiplicity,~sep~color,~tikz~
11329   and~total~width~(the~latter~is~meaningful~only~in~cunjunction~with~'tikz').
11330 }

```

If fact, there is also the key dotted but it won't be very useful since we provide \hdottedline, \cdottedline and the letter :.

```

11331 \@@_msg_new:nnn { Unknown~key~for~Block }
11332 {
11333   Unknown~key. \\
11334   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11335   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11336   \c_@@_available_keys_str
11337 }
11338 {
11339   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11340   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11341   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11342   transparent~and~vlines.
11343 }
11344 \@@_msg_new:nnn { Unknown~key~for~Brace }
11345 {
11346   Unknown~key.\\
11347   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11348   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11349   Your~key~will~be~ignored. \\
11350   \c_@@_available_keys_str
11351 }
11352 {
11353   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11354   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11355   right~shorten)~and~yshift.
11356 }

```

```

11357 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11358 {
11359   Unknown~key.\\
11360   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11361   \c_@@_available_keys_str
11362 }
11363 {
11364   The~available~keys~are~(in~alphabetic~order):~
11365   delimiters/color,~
11366   rules~(with~the~subkeys~'color'~and~'width'),~
11367   sub-matrix~(several~subkeys)~
11368   and~xdots~(several~subkeys).~
11369   The~latter~is~for~the~command~ \token_to_str:N \line .
11370 }
11371 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11372 {
11373   Unknown~key.\\
11374   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11375   \c_@@_available_keys_str
11376 }
11377 {
11378   The~available~keys~are~(in~alphabetic~order):~
11379   create-cell-nodes,~
11380   delimiters/color~and~
11381   sub-matrix~(several~subkeys).
11382 }
11383 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11384 {
11385   Unknown~key.\\
11386   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11387   \c_@@_available_keys_str
11388 }
11389 {
11390   The~available~keys~are~(in~alphabetic~order):~
11391   'delimiters/color',~
11392   'extra-height',~
11393   'hlines',~
11394   'hvlines',~
11395   'left-xshift',~
11396   'name',~
11397   'right-xshift',~
11398   'rules'~(with~the~subkeys~'color'~and~'width'),~
11399   'slim',~
11400   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11401 }
11402 \@@_msg_new:nnn { Unknown~key~for~notes }
11403 {
11404   Unknown~key.\\
11405   The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11406   \c_@@_available_keys_str
11407 }
11408 {
11409   The~available~keys~are~(in~alphabetic~order):~
11410   bottomrule,~
11411   code-after,~
11412   code-before(+),~
11413   detect-duplicates,~
11414   enumitem-keys,~
11415   enumitem-keys-para,~
11416   para,~
11417   label-in-list,~
11418   label-in-tabular~and~
11419   style.

```



```

11420 }
11421 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11422 {
11423   Unknown~key.\\
11424   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11425   \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11426   \c_@@_available_keys_str
11427 }
11428 {
11429   The~available~keys~are~(in~alphabetic~order):~
11430   bold,~
11431   cell-space-top-limit(+),~
11432   cell-space-bottom-limit(+),~
11433   cell-space-limits(+),~
11434   color,~
11435   fill~(alias:~rowcolor),~
11436   nb-rows,~
11437   opacity~and~
11438   rounded-corners.
11439 }
11440 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11441 {
11442   Unknown~key.\\
11443   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11444   \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11445   \c_@@_available_keys_str
11446 }
11447 {
11448   The~available~keys~are~(in~alphabetic~order):~
11449   &~in~blocks,~
11450   allow-duplicate-names,~
11451   ampersand-in-blocks,~
11452   caption-above,~
11453   cell-space-bottom-limit(+),~
11454   cell-space-limits(+),~
11455   cell-space-top-limit(+),~
11456   code-for-first-col(+),~
11457   code-for-first-row(+),~
11458   code-for-last-col(+),~
11459   code-for-last-row(+),~
11460   corners,~
11461   custom-key,~
11462   create-extra-nodes,~
11463   create-medium-nodes,~
11464   create-large-nodes,~
11465   custom-line,~
11466   delimiters~(several~subkeys),~
11467   end-of-row,~
11468   first-col,~
11469   first-row,~
11470   h(v)lines,~
11471   h(v)lines-except-borders,~
11472   last-col,~
11473   last-row,~
11474   left-margin,~
11475   light-syntax,~
11476   light-syntax-expanded,~
11477   matrix/columns-type,~
11478   no-cell-nodes,~
11479   notes~(several~subkeys),~
11480   nullify-dots,~
11481   pgf-node-code,~
11482   renew-dots,~

```

```

11483   renew-matrix,~
11484   respect-arraystretch,~
11485   rounded-corners,~
11486   right-margin,~
11487   rules~(with~the~subkeys~'color'~and~'width'),~
11488   small,~
11489   sub-matrix~(several~subkeys),~
11490   vl原因,~
11491   xdots~(several~subkeys).
11492 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11493 \@@_msg_new:nnn { Unknown-key-for~NiceArray }
11494 {
11495   Unknown-key.\\
11496   The-key~' \l_keys_key_str '~is~unknown~for~the~environment~
11497   \{NiceArray\}~.~Your~key~will~be~ignored. \\
11498   \c_@@_available_keys_str
11499 }
11500 {
11501   The-available-keys~are~(in~alphabetic~order):~
11502   &-in-blocks,~
11503   ampersand-in-blocks,~
11504   b,~
11505   baseline,~
11506   c,~
11507   cell-space-bottom-limit,~
11508   cell-space-limits,~
11509   cell-space-top-limit,~
11510   code-after,~
11511   code-for-first-col(+),~
11512   code-for-first-row(+),~
11513   code-for-last-col(+),~
11514   code-for-last-row(+),~
11515   columns-width,~
11516   corners,~
11517   create-blocks-in-col,~
11518   create-extra-nodes,~
11519   create-medium-nodes,~
11520   create-large-nodes,~
11521   draw-trees-in-col,~
11522   extra-left-margin,~
11523   extra-right-margin,~
11524   first-col,~
11525   first-row,~
11526   h(v)lines,~
11527   h(v)lines-except-borders,~
11528   last-col,~
11529   last-row,~
11530   left-margin,~
11531   light-syntax,~
11532   light-syntax-expanded,~
11533   name,~
11534   no-cell-nodes,~
11535   nullify-dots,~
11536   pgf-node-code,~
11537   renew-dots,~
11538   respect-arraystretch,~
11539   right-margin,~
11540   rounded-corners,~
11541   rules~(with~the~subkeys~'color'~and~'width'),~
11542   small,~
11543   t,~

```

```

11544     vlines,~
11545     xdots/color,~
11546     xdots/shorten-start(+),~
11547     xdots/shorten-end(+),~
11548     xdots/shorten(+)-and~
11549     xdots/line-style.
11550 }

```

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`).

```

11551 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11552 {
11553   Unknown~key.\\
11554   The~key~' \l_keys_key_str '~is~unknown~for~the~
11555   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11556   \c_@@_available_keys_str
11557 }
11558 {
11559   The~available~keys~are~(in~alphabetic~order):~
11560   &~in~blocks,~
11561   ampersand~in~blocks,~
11562   b,~
11563   baseline,~
11564   c,~
11565   cell-space-bottom-limit,~
11566   cell-space-limits,~
11567   cell-space-top-limit,~
11568   code-after,~
11569   code-for-first-col(+),~
11570   code-for-first-row(+),~
11571   code-for-last-col(+),~
11572   code-for-last-row(+),~
11573   columns-type,~
11574   columns-width,~
11575   corners,~
11576   create-blocks-in-col,~
11577   create-extra-nodes,~
11578   create-medium-nodes,~
11579   create-large-nodes,~
11580   draw-trees-in-col,~
11581   extra-left-margin,~
11582   extra-right-margin,~
11583   first-col,~
11584   first-row,~
11585   h(v)lines,~
11586   h(v)lines-except-borders,~
11587   l,~
11588   last-col,~
11589   last-row,~
11590   left-margin,~
11591   light-syntax,~
11592   light-syntax-expanded,~
11593   name,~
11594   no-cell-nodes,~
11595   nullify-dots,~
11596   pgf-node-code,~
11597   r,~
11598   renew-dots,~
11599   respect-arraystretch,~
11600   right-margin,~
11601   rounded-corners,~
11602   rules~(with~the~subkeys~'color'~and~'width'),~
11603   small,~

```

```

11604     t,~
11605     vlines,~
11606     xdots/color,~
11607     xdots/shorten-start(+),~
11608     xdots/shorten-end(+),~
11609     xdots/shorten(+)-and~
11610     xdots/line-style.
11611 }

11612 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11613 {
11614     Unknown~key.\\
11615     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11616     \{NiceTabular\}.~Your~key~will~be~ignored. \\
11617     \c_@@_available_keys_str
11618 }
11619 {
11620     The~available~keys~are~(in~alphabetic~order):~
11621     &in~blocks,~
11622     ampersand~in~blocks,~
11623     b,~
11624     baseline,~
11625     c,~
11626     caption,~
11627     cell-space-bottom-limit,~
11628     cell-space-limits,~
11629     cell-space-top-limit,~
11630     code-after,~
11631     code-for-first-col(+),~
11632     code-for-first-row(+),~
11633     code-for-last-col(+),~
11634     code-for-last-row(+),~
11635     columns-width,~
11636     corners,~
11637     custom-line,~
11638     create-blocks-in-col,~
11639     create-extra-nodes,~
11640     create-medium-nodes,~
11641     create-large-nodes,~
11642     draw-trees-in-col,~
11643     extra-left-margin,~
11644     extra-right-margin,~
11645     first-col,~
11646     first-row,~
11647     h(v)lines,~
11648     h(v)lines-except-borders,~
11649     label,~
11650     last-col,~
11651     last-row,~
11652     left-margin,~
11653     light-syntax,~
11654     light-syntax-expanded,~
11655     name,~
11656     no-cell-nodes,~
11657     notes~(several~subkeys),~
11658     nullify-dots,~
11659     pgf-node-code,~
11660     renew-dots,~
11661     respect-arraystretch,~
11662     right-margin,~
11663     rounded-corners,~
11664     rules~(with~the~subkeys~'color'~and~'width'),~
11665     short-caption,~
11666     t,~

```

```

11667     tabularnote,~
11668     vlines,~
11669     xdots/color,~
11670     xdots/shorten-start(+),~
11671     xdots/shorten-end(+),~
11672     xdots/shorten(+)-and~
11673     xdots/line-style.
11674 }

11675 \@@_msg_new:nnn { Duplicate~name }
11676 {
11677     Duplicate~name.\\
11678     The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
11679     the~same~environment~name~twice.~You~can~go~on,~but,~
11680     maybe,~you~will~have~incorrect~results~especially~
11681     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11682     message~again,~use~the~key~'allow-duplicate-names'~in~
11683     ' \token_to_str:N \NiceMatrixOptions '.\\
11684     \bool_if:NF \g_@@_messages_for_Overleaf_bool
11685     { For~a~list~of~the~names~already~used,~type-H~<return>. }
11686 }
11687 {
11688     The~names~already~defined~in~this~document~are:~
11689     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11690 }

11691 \@@_msg_new:nn { caption-above~in~env }
11692 {
11693     The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.~
11694     Your~key~will~be~ignored.
11695 }

11696 \@@_msg_new:nn { show-cell-names }
11697 {
11698     There~is~no~key~'show-cell-names'~in~nicematrix.\\
11699     You~should~use~the~command~\token_to_str:N \ShowCellNames\
11700     in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11701     \CodeAfter.~Your~key~will~be~ignored.
11702 }

11703 \@@_msg_new:nn { Option~auto~for~columns-width }
11704 {
11705     Erroneous~use.\\
11706     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11707     Your~key~will~be~ignored.
11708 }

11709 \@@_msg_new:nn { NiceTabularX~without~X }
11710 {
11711     NiceTabularX~without~X.\\
11712     You~should~not~use~\{NiceTabularX\}~without~X~columns.~However,~you~can~go~on.
11713 }

11714 \@@_msg_new:nn { Preamble~forgotten }
11715 {
11716     Preamble~forgotten.\\
11717     You~have~probably~forgotten~the~preamble~of~your~
11718     \@@_full_name_env: .
11719 }

11720 \@@_msg_new:nn { Invalid~col~number }
11721 {
11722     Invalid~column~number.\\
11723     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11724     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
11725     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11726 }

```

```

11727 \@@_msg_new:nn { Invalid~row~number }
11728 {
11729   Invalid~row~number.\\
11730   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11731   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
11732   Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11733 }

11734 \@@_define_com:NNN p ( )
11735 \@@_define_com:NNN b [ ]
11736 \@@_define_com:NNN v | |
11737 \@@_define_com:NNN V \l \l
11738 \@@_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>	38
9	The <code>\CodeBefore</code>	53
10	The environment <code>{NiceArrayWithDelims}</code>	57
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	80
13	The environment <code>{NiceMatrix}</code> and its variants	97
	13.1 Definition of <code>{pNiceMatrix}</code> . . . . .	97
	13.2 The key <code>renew-matrix</code> . . . . .	98
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	98
15	After the construction of the array	100
16	We draw the dotted lines	107
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	124
18	User commands available in the new environments	130
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	136
20	The command <code>\RowStyle</code>	138
21	Colors of cells, rows and columns	141
22	The vertical and horizontal rules	153
23	The empty corners	175
24	The environment <code>{NiceMatrixBlock}</code>	179
25	The extra nodes	180
26	The blocks	185
27	Automatic arrays	213
28	The redefinition of the command <code>\dotfill</code>	214
29	The command <code>\diagbox</code>	214

30	The keyword <code>\CodeAfter</code>	216
31	The delimiters in the preamble	216
32	The command <code>\SubMatrix</code>	218
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	227
34	The commands <code>HBrace</code> et <code>VBrace</code>	230
35	The command <code>TikzEveryCell</code>	233
36	The key <code>draw-trees-in-col</code>	234
37	The key <code>create-blocks-in-col</code>	236
38	The command <code>\ShowCellNames</code>	236
39	We process the options at package loading	238
40	About the package underscore	240
41	Compatibility with <code>threeparttable</code>	240
42	Error messages of the package	240