

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

July 14, 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 7.11 of `nicematrix`, at the date of 2026/07/14.

```

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`¹

```

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 }

```

¹See 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 parallelized². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```
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 Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```
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
```

²It’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”.³

```
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 }
```

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

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`.⁴
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\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.

⁴More 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 cur-ryfication.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

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

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

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

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

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

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

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

```
654       \int_gincr:N \g_@@_tabularnote_int
655       \refstepcounter { tabularnote }
656       \int_compare:nNnT \l_tmpa_int = \c@tabularnote
657       { \int_gincr:N \c@tabularnote }
658       \seq_clear:N \l_@@_notes_labels_seq
659       \bool_lazy_or:nnTF
660       { \str_if_eq_p:ee 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

```

The following parameter corresponds to the key `width-of-false`.

```

775 \dim_new:N \l_@@_width_of_false_dim
776 \dim_set:Nn \l_@@_width_of_false_dim { 3 pt }

```

```

777 \keys_define:nn { nicematrix / xdots }
778 {

```

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.).

```

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

```

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

```

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

```

```
835 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).

```
836 draw-first .code:n = \prg_do_nothing: ,
837 unknown .code:n =
838   \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
839 }
```

```
840 \keys_define:nn { nicematrix / trees }
841 {
842   color.tl_set:N = \l_@@_trees_color_tl ,
843   color .value_required:n = true ,
844   line-width .dim_set:N = \l_@@_trees_line_width_dim ,
845   rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
846   rounded-corners .default:n = 2 pt ,
847   unknown .code:n =
848     \@@_unknown_key:nn { nicematrix / trees } { Unknown~key~for~trees }
849 }
```

```
850 \keys_define:nn { nicematrix / rules }
851 {
852   fix-vertex .code:n =
853     \bool_set_true:N \l_@@_fix_vertex_bool
854     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
855     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
856   color .tl_set:N = \l_@@_rules_color_tl ,
857   color .value_required:n = true ,
858   width .dim_set:N = \arrayrulewidth ,
859   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
860 }
```

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

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

```

887     }
888   } ,
889   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.

```

890   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
891   rounded-corners .default:n = 4 pt ,
892   custom-line .code:n = \@@_custom_line:n { #1 } ,
893   default-line .code:n = \@@_default_line:n { #1 } ,
894   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
895   rules .value_required:n = true ,
896   trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
897   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`.

```

898   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
899   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
900   cell-space-top-limit+ .code:n =
901     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
902   cell-space-top-limit+ .value_required:n = true ,
903   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
904   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
905   cell-space-bottom-limit+ .code:n =
906     \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
907   cell-space-bottom-limit+ .value_required:n = true ,
908   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
909   cell-space-limits .meta:n =
910     {
911       cell-space-top-limit = #1 ,
912       cell-space-bottom-limit = #1 ,
913     } ,
914   cell-space-limits .value_required:n = true ,
915   cell-space-limits+ .meta:n =
916     {
917       cell-space-top-limit += #1 ,
918       cell-space-bottom-limit += #1 ,
919     } ,
920   cell-space-limits+ .value_required:n = true ,
921   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
922   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
923   light-syntax .choices:nn = { true , false }
924   {
925     \tl_if_eq:VnTF \l_keys_value_tl { true }
926     { \bool_set_true:N \l_@@_light_syntax_bool }
927     { \bool_set_false:N \l_@@_light_syntax_bool }
928     \bool_set_false:N \l_@@_light_syntax_expanded_bool
929   } ,
930   light-syntax .default:n = true ,
931   light-syntax-expanded .choices:nn = { true , false }
932   {
933     \tl_if_eq:VnTF \l_keys_value_tl { true }
934     {
935       \bool_set_true:N \l_@@_light_syntax_bool
936       \bool_set_true:N \l_@@_light_syntax_expanded_bool
937     }
938     {
939       \bool_set_false:N \l_@@_light_syntax_bool
940       \bool_set_false:N \l_@@_light_syntax_expanded_bool
941     }
942   } ,
943   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.

```

944     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
945     end-of-row .value_required:n = true ,
946     end-of-row .initial:n = ; ,
947
948     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
949     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
950     last-row .int_set:N = \l_@@_last_row_int ,
951     last-row .default:n = -1 ,
952
953     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
954     code-for-first-col .value_required:n = true ,
955     code-for-first-col+ .code:n =
956       { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
957     code-for-first-col+ .value_required:n = true ,
958     code-for-first-col+ .meta:n = { code-for-first-col+ = #1 } ,
959
960     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
961     code-for-last-col .value_required:n = true ,
962     code-for-last-col+ .code:n =
963       { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
964     code-for-last-col+ .value_required:n = true ,
965     code-for-last-col+ .meta:n = { code-for-last-col+ = #1 } ,
966
967     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
968     code-for-first-row .value_required:n = true ,
969     code-for-first-row+ .code:n =
970       { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
971     code-for-first-row+ .value_required:n = true ,
972     code-for-first-row+ .meta:n = { code-for-first-row+ = #1 } ,
973
974     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
975     code-for-last-row .value_required:n = true ,
976     code-for-last-row+ .code:n =
977       { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
978     code-for-last-row+ .value_required:n = true ,
979     code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
980
981     hlines .clist_set:N = \l_@@_hlines_clist ,
982     hlines .default:n = all ,
983     vlines .clist_set:N = \l_@@_vlines_clist ,
984     vlines .default:n = all ,
985
986     vlines-in-sub-matrix .code:n =
987       {
988         \tl_if_single_token:nTF { #1 }
989         {
990           \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
991             { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

992       { \cs_set_eq:cN { @@_#1 : } \@@_make_preamble_vlism:n }
993     }
994     { \@@_error:n { One-letter-allowed } }
995   } ,
996   vlines-in-sub-matrix .value_required:n = true ,
997   hvlines .code:n =
998     {
999       \bool_set_true:N \l_@@_hvlines_bool
1000       \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1001       \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1002     } ,
1003   hvlines .value_forbidden:n = true ,

```

```

1004   hvlines-except-borders .code:n =
1005   {
1006       \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1007       \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1008       \bool_set_true:N \l_@@_hvlines_bool
1009       \bool_set_true:N \l_@@_except_borders_bool
1010   } ,
1011   hlines-except-borders .value_forbidden:n = true ,
1012   hlines-except-borders .code:n =
1013   {
1014       \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1015       \bool_set_true:N \l_@@_hlines_bool
1016       \bool_set_true:N \l_@@_except_borders_bool
1017   } ,
1018   hlines-except-borders .value_forbidden:n = true ,
1019   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1020   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.

```

1021   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1022   renew-dots .value_forbidden:n = true ,
1023   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1024   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1025   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1026   create-extra-nodes .meta:n =
1027   { create-medium-nodes , create-large-nodes } ,
1028   left-margin .dim_set:N = \l_@@_left_margin_dim ,
1029   left-margin .default:n = \arraycolsep ,
1030   right-margin .dim_set:N = \l_@@_right_margin_dim ,
1031   right-margin .default:n = \arraycolsep ,
1032   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1033   margin .default:n = \arraycolsep ,
1034   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1035   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1036   extra-margin .meta:n =
1037   { extra-left-margin = #1 , extra-right-margin = #1 } ,
1038   extra-margin .value_required:n = true ,
1039   respect-arraystretch .code:n =
1040   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1041   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.

```

1042   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1043   pgf-node-code .value_required:n = true ,
1044   }

```

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

```

1045   \keys_define:nn { nicematrix / environments }
1046   {
1047       draw-trees-in-col .code:n =
1048       \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1049       draw-trees-in-col .value_required:n = true ,
1050       create-blocks-in-col .code:n =
1051       \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1052       create-blocks-in-col .value_required:n = true ,
1053       corners .clist_set:N = \l_@@_corners_clist ,
1054       corners .default:n = { NW , SW , NE , SE } ,
1055       code-before .code:n =
1056       {

```

```

1057 \tl_if_empty:nF { #1 }
1058 {
1059     \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1060     \bool_set_true:N \l_@@_code_before_bool
1061 }
1062 } ,
1063 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}`.

```

1064 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1065 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1066 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).

```

1067 baseline .tl_set:N = \l_@@_baseline_tl ,
1068 baseline .value_required:n = true ,
1069 baseline .initial:n = c ,
1070 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)`.

```

1071 \str_if_eq:eeTF { #1 } { auto }
1072 { \bool_set_true:N \l_@@_auto_columns_width_bool }
1073 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1074 columns-width .value_required:n = true ,
1075 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.

```

1076 \legacy_if:nF { measuring@ }
1077 {
1078     \str_set:Nn \l_@@_name_str { #1 }
1079     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1080     { \@@_err_duplicate_names:n { #1 } }
1081     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1082 } ,
1083 name .value_required:n = true ,
1084 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1085 code-after .value_required:n = true ,
1086 }

1087 \cs_set:Npn \@@_err_duplicate_names:n #1
1088 { \@@_error:nn { Duplicate-name } { #1 } }

1089 \keys_define:nn { nicematrix / notes }
1090 {
1091     no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1092     para .bool_set:N = \l_@@_notes_para_bool ,
1093     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1094     code-before .value_required:n = true ,
1095     code-before+ .code:n =
1096         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1097     code-before+ .value_required:n = true ,
1098     code-before~+ .code:n =
1099         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1100     code-before~+ .value_required:n = true ,
1101     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1102     code-after .value_required:n = true ,
1103     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1104     style .cs_set:Np = \@@_notes_style:n #1 ,
1105     style .value_required:n = true ,
1106     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,

```

```

1107 label-in-tabular .value_required:n = true ,
1108 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1109 label-in-list .value_required:n = true ,
1110 enumitem-keys .code:n =
1111 {
1112   \AtBeginDocument
1113   {
1114     \IfPackageLoadedT { enumitem }
1115     { \setlist* [ tabularnotes ] { #1 } }
1116   }
1117 } ,
1118 enumitem-keys .value_required:n = true ,
1119 enumitem-keys-para .code:n =
1120 {
1121   \AtBeginDocument
1122   {
1123     \IfPackageLoadedT { enumitem }
1124     { \setlist* [ tabularnotes* ] { #1 } }
1125   }
1126 } ,
1127 enumitem-keys-para .value_required:n = true ,
1128 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1129 unknown .code:n =
1130   \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1131 }

```

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

```

1132 \keys_define:nn { nicematrix / delimiters }
1133 {
1134   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1135   color .tl_set:N = \l_@@_delimiters_color_tl ,
1136   color .value_required:n = true ,
1137 }

```

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

```

1138 \keys_define:nn { nicematrix }
1139 {
1140   NiceMatrixOptions .inherit:n =
1141     { nicematrix / Global } ,
1142   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1143   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1144   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1145   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1146   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1147   SubMatrix / rules .inherit:n = nicematrix / rules ,
1148   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1149   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1150   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1151   NiceMatrix .inherit:n =
1152     {
1153       nicematrix / Global ,
1154       nicematrix / environments ,
1155     } ,
1156   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1157   NiceMatrix / rules .inherit:n = nicematrix / rules ,

```

```

1158 NiceMatrix / trees .inherit:n = nicematrix / trees ,
1159 NiceTabular .inherit:n =
1160 {
1161     nicematrix / Global ,
1162     nicematrix / environments
1163 } ,
1164 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1165 NiceTabular / rules .inherit:n = nicematrix / rules ,
1166 NiceTabular / notes .inherit:n = nicematrix / notes ,
1167 NiceTabular / trees .inherit:n = nicematrix / trees ,
1168 NiceArray .inherit:n =
1169 {
1170     nicematrix / Global ,
1171     nicematrix / environments ,
1172 } ,
1173 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1174 NiceArray / rules .inherit:n = nicematrix / rules ,
1175 NiceArray / trees .inherit:n = nicematrix / trees ,
1176 pNiceArray .inherit:n =
1177 {
1178     nicematrix / Global ,
1179     nicematrix / environments ,
1180 } ,
1181 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1182 pNiceArray / rules .inherit:n = nicematrix / rules ,
1183 pNiceArray / trees .inherit:n = nicematrix / trees
1184 }

```

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

```

1185 \keys_define:nn { nicematrix / NiceMatrixOptions }
1186 {
1187     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1188     delimiters / color .value_required:n = true ,
1189     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1190     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1191     delimiters .value_required:n = true ,
1192     width .dim_set:N = \l_@@_width_dim ,
1193     last-col .code:n =
1194     \tl_if_empty:nF { #1 }
1195     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1196     \int_zero:N \l_@@_last_col_int ,
1197     small .bool_set:N = \l_@@_small_bool ,
1198     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.

```

1199 renew-matrix .code:n = \@@_renew_matrix: ,
1200 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}`.

```

1201 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.

```

1202 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)`.

```

1203 \str_if_eq:eeTF { #1 } { auto }
1204 { \@@_error:n { Option~auto~for~columns-width } }
1205 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1206   allow-duplicate-names .code:n =
1207     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1208   allow-duplicate-names .value_forbidden:n = true ,
1209   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1210   notes .value_required:n = true ,
1211   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1212   sub-matrix .value_required:n = true ,
1213   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1214   matrix / columns-type .value_required:n = true ,
1215   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1216   unknown .code:n =
1217     \@@_unknown_key:nn
1218       { nicematrix / Global , nicematrix / NiceMatrixOptions }
1219       { Unknown-key-for-NiceMatrixOptions }
1220 }

```

`\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.

```

1221 \NewDocumentCommand \NiceMatrixOptions { m }
1222 { \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.

```

1223 \keys_define:nn { nicematrix / NiceMatrix }
1224 {
1225   last-col .code:n = \tl_if_empty:nTF { #1 }
1226     {
1227       \bool_set_true:N \l_@@_last_col_without_value_bool
1228       \int_set:Nn \l_@@_last_col_int { -1 }
1229     }
1230     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1231   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1232   columns-type .value_required:n = true ,
1233   l .meta:n = { columns-type = l } ,
1234   r .meta:n = { columns-type = r } ,
1235   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1236   delimiters / color .value_required:n = true ,
1237   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1238   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1239   delimiters .value_required:n = true ,
1240   small .bool_set:N = \l_@@_small_bool ,
1241   small .value_forbidden:n = true ,
1242   unknown .code:n =
1243     \@@_unknown_key:nn
1244       { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1245       { Unknown-key-for-NiceMatrix }
1246 }

```

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

```

1247 \keys_define:nn { nicematrix / NiceArray }
1248 {

```

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.

```

1249   small .bool_set:N = \l_@@_small_bool ,
1250   small .value_forbidden:n = true ,
1251   last-col .code:n = \tl_if_empty:nF { #1 }

```

```

1252             { \@@_error:n { last-col-non-empty-for-NiceArray } }
1253             \int_zero:N \l_@@_last_col_int ,
1254     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1255     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1256     unknown .code:n =
1257       \@@_unknown_key:nn
1258       { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
1259       { Unknown-key-for-NiceArray }
1260   }
1261 \keys_define:nn { nicematrix / pNiceArray }
1262 {
1263   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1264   last-col .code:n = \tl_if_empty:nF { #1 }
1265     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1266     \int_zero:N \l_@@_last_col_int ,
1267   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1268   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1269   delimiters / color .value_required:n = true ,
1270   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1271   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1272   delimiters .value_required:n = true ,
1273   small .bool_set:N = \l_@@_small_bool ,
1274   small .value_forbidden:n = true ,
1275   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1276   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1277   unknown .code:n =
1278     \@@_unknown_key:nn
1279     { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1280     { Unknown-key-for-NiceMatrix }
1281 }

```

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

```

1282 \keys_define:nn { nicematrix / NiceTabular }
1283 {

```

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.

```

1284   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1285   \bool_set_true:N \l_@@_width_used_bool ,
1286   width .value_required:n = true ,
1287   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1288   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1289   tabularnote .value_required:n = true ,
1290   caption .tl_set:N = \l_@@_caption_tl ,
1291   caption .value_required:n = true ,
1292   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1293   short-caption .value_required:n = true ,
1294   label .tl_set:N = \l_@@_label_tl ,
1295   label .value_required:n = true ,
1296   last-col .code:n = \tl_if_empty:nF { #1 }
1297     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1298     \int_zero:N \l_@@_last_col_int ,
1299   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1300   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1301   unknown .code:n =
1302     \@@_unknown_key:nn
1303     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1304     { Unknown-key-for-NiceTabular }
1305 }

```

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
```

```
1306 \keys_define:nn { nicematrix / CodeAfter }
1307 {
1308   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1309   delimiters / color .value_required:n = true ,
1310   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1311   rules .value_required:n = true ,
1312   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1313   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1314   sub-matrix .value_required:n = true ,
1315   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1316 }
```

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}`).

```
1317 \cs_new_protected:Npn \@@_cell_begin:
1318 {
```

`\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.

```
1319   \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).

```
1320   \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.

```
1321   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1322   \int_gincr:N \c@jCol
```

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

Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```
1323   \if_int_compare:w \c@jCol = \c_one_int
1324     \if_int_compare:w \l_@@_first_col_int = \c_one_int
1325       \@@_begin_of_row:
1326     \fi:
1327   \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:`.

```
1328   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1329 \@@_tuning_not_tabular_begin:
1330 \@@_tuning_exterior_rows:
1331 \g_@@_row_style_tl
1332 }
1333 \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.

```

1334 \cs_new_protected:Npn \@@_tuning_first_last_row:
1335 {
1336   \if_int_compare:w \c@iRow = \c_zero_int
1337   \if_int_compare:w \c@jCol > \c_zero_int
1338   \l_@@_code_for_first_row_tl
1339   \@@_tuning_key_small:
1340   \xglobal \colorlet { nicematrix-first-row } { . }
1341   \fi:
1342   \else:
1343   \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1344   \fi:
1345 }

```

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.

```

1346 \cs_new_protected:Npn \@@_tuning_last_row:
1347 {
1348   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1349   \l_@@_code_for_last_row_tl
1350   \@@_tuning_key_small:
1351   \xglobal \colorlet { nicematrix-last-row } { . }
1352   \fi:
1353 }

```

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

```

1354 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1355 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1356 {
1357   \m@th
1358   $ % $

```

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

```

1359 \@@_tuning_key_small:
1360 }
1361 \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.

```

1362 \cs_new_protected:Npn \@@_begin_of_row:
1363 {
1364   \int_gincr:N \c@iRow
1365   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1366   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1367   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1368   \pgfpicture
1369   \pgfrememberpicturepositiononpagetrue
1370   \pgfcoordinate
1371     { \@@_env: - row - \int_use:N \c@iRow - base }
1372     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1373   \str_if_empty:NF \l_@@_name_str
1374     {
1375       \pgfnodealias
1376         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1377         { \@@_env: - row - \int_use:N \c@iRow - base }
1378     }
1379   \endpgfpicture
1380 }

```

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.

```

1381 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1382 {
1383   \if_int_compare:w \c@iRow = \c_zero_int
1384     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1385       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box

```

```

1386 \fi:
1387 \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1388 \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1389 \fi:
1390 \else:
1391 \if_int_compare:w \c@iRow = \c_one_int
1392 \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1393 \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1394 \fi:
1395 \fi:
1396 \fi:
1397 }

1398 \cs_new_protected:Npn \@@_rotate_cell_box:
1399 {
1400 \box_rotate:Nn \l_@@_cell_box
1401 { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1402 \bool_if:NTF \g_@@_rotate_c_bool
1403 {
1404 \hbox_set:Nn \l_@@_cell_box
1405 {
1406 \IfFormatAtLeastTF { 2026-04-01 }
1407 { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1408 {
1409 \m@th
1410 $ % $
1411 \vcenter { \box_use:N \l_@@_cell_box }
1412 $ % $
1413 }
1414 }
1415 }
1416 {
1417 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1418 {
1419 \vbox_set_top:Nn \l_@@_cell_box
1420 {
1421 \vbox_to_zero:n { }
1422 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1423 \box_use:N \l_@@_cell_box
1424 }
1425 }
1426 }
1427 \bool_gset_false:N \g_@@_rotate_bool
1428 \bool_gset_false:N \g_@@_rotate_c_bool
1429 \bool_gset_false:N \g_@@_rotate_minus_bool
1430 }

```

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.

```

1431 \cs_set_protected:Npn \@@_adjust_size_box:
1432 {
1433   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1434     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1435       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1436     \fi:
1437     \global \g_@@_blocks_wd_dim = \c_zero_dim
1438   \fi:
1439   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1440     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1441       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1442     \fi:
1443     \global \g_@@_blocks_dp_dim = \c_zero_dim
1444   \fi:
1445   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1446     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1447       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1448     \fi:
1449     \global \g_@@_blocks_ht_dim = \c_zero_dim
1450   \fi:
1451 }
1452 \cs_new_protected:Npn \@@_cell_end:
1453 {

```

The following command is nullified in the tabulars.

```

1454   \@@_tuning_not_tabular_end:
1455   \hbox_set_end:
1456   \@@_cell_end_i:
1457 }

```

```

\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.

```

1458 \cs_new_protected:Npn \l_@@_cell_end_i:
1459 {

```

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.

```

1460 \g_@@_cell_after_hook_tl
1461 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1462 \@@_adjust_size_box:
1463 \box_set_ht:Nn \l_@@_cell_box
1464 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1465 \box_set_dp:Nn \l_@@_cell_box
1466 { \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”).

```

1467 \@@_update_max_cell_width:

```

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

```

1468 \@@_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.

```

1469 \bool_if:NTF \g_@@_empty_cell_bool
1470 { \box_use_drop:N \l_@@_cell_box }
1471 {
1472 \bool_if:NTF \g_@@_not_empty_cell_bool
1473 \@@_print_node_cell:
1474 {
1475 \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1476 \@@_print_node_cell:
1477 \else:
1478 \box_use_drop:N \l_@@_cell_box
1479 \fi:
1480 }
1481 }
1482 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1483 \global \g_@@_col_total_int = \c@jCol
1484 \fi:
1485 \global \let \g_@@_empty_cell_bool \c_false_bool
1486 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1487 }

```

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:

```
1488 \cs_new_protected:Npn \@@_update_max_cell_width:
1489 {
1490   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1491     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1492   \fi:
1493 }
```

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`).

```
1494 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1495 {
1496   \@@_math_toggle:
1497   \hbox_set_end:
1498   \bool_if:NF \g_@@_rotate_bool
1499   {
1500     \hbox_set:Nn \l_@@_cell_box
1501     {
1502       \makebox [ \l_@@_col_width_dim ] [ s ]
1503       { \hbox_unpack_drop:N \l_@@_cell_box }
1504     }
1505   }
1506   \@@_cell_end_i:
1507 }
```

```
1508 \pgfset
1509 {
1510   nicematrix / cell-node /.style =
1511   {
1512     inner~sep = \c_zero_dim ,
1513     minimum~width = \c_zero_dim
1514   }
1515 }
```

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.

```
1516 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1517 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1518 {
1519   \use:c
1520   {
1521     __siunitx_table_align_
1522     \bool_if:NTF \l__siunitx_table_text_bool
1523       \l__siunitx_table_align_text_tl
1524       \l__siunitx_table_align_number_str
1525     :n
1526   }
1527   { #1 }
1528 }
```

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!

```

1529 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1530 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1531 {
1532   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1533   \hbox:n
1534   {
1535     \pgfsys@markposition
1536     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1537   }
1538   #1
1539   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1540   \hbox:n
1541   {
1542     \pgfsys@markposition
1543     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1544   }
1545 }

1546 \cs_new_protected:Npn \@@_print_node_cell:
1547 {
1548   \socket_use:nn { nicematrix / siunitx-wrap }
1549   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1550 }

```

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

```

1551 \cs_new_protected:Npn \@@_node_cell_active:
1552 {
1553   \pgfpicture
1554   \pgfsetbaseline \c_zero_dim
1555   \pgfrememberpicturepositiononpagetrue
1556   \pgfset { nicematrix / cell-node }
1557   \pgfnode
1558   { rectangle }
1559   { base }
1560   {

```

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).

```

1561   \sys_if_engine_xetex:T { \set@color }
1562   \box_use:N \l_@@_cell_box
1563 }
1564 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1565 { \l_@@_pgf_node_code_tl }
1566 \str_if_empty:NF \l_@@_name_str
1567 {
1568   \pgfnodealias
1569   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1570   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1571 }
1572 \endpgfpicture
1573 }
1574 \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1575 \cs_new_protected:Npn \@@_node_cell_inactive:
1576 { \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).

```
1577 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1578 {
1579   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1580   { g_@@_ #2 _ lines _ tl }
1581   {
1582     \use:c { @@ _ draw _ #2 : nnn }
1583     { \int_use:N \c@iRow }
1584     { \int_use:N \c@jCol }
1585     { \exp_not:n { #3 } }
1586   }
1587 }

1588 \cs_new_protected:Npn \@@_array:n
1589 {
1590   \dim_set:Nn \col@sep
1591   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1592   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1593   { \def \@halignto { } }
1594   { \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`.

```
1595   \@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).

1596   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1597   }
1598 \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`.

```
1599 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1600 \cs_new_protected:Npn \@@_create_row_node:
1601 {
1602   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1603   {
1604     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1605     \@@_create_row_node_i:
1606   }
1607 }
```

```

1608 \cs_new_protected:Npn \@@_create_row_node_i:
1609 {

```

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

```

1610   \hbox
1611   {
1612     \bool_if:NT \l_@@_code_before_bool
1613     {
1614       \vtop
1615       {
1616         \skip_vertical:N 0.5\arrayrulewidth
1617         \pgfsys@markposition
1618         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1619         \skip_vertical:N -0.5\arrayrulewidth
1620       }
1621     }
1622     \pgfpicture
1623     \pgfrememberpicturepositiononpagetrue
1624     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1625     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1626     \str_if_empty:NF \l_@@_name_str
1627     {
1628       \pgfnodealias
1629       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1630       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1631     }
1632     \endpgfpicture
1633   }
1634 }

```

```

1635 \cs_new_protected:Npn \@@_in_everycr:
1636 {
1637   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1638   \tbl_update_cell_data_for_next_row:
1639   \int_gzero:N \c@jCol
1640   \bool_gset_false:N \g_@@_after_col_zero_bool
1641   \bool_if:NF \g_@@_row_of_col_done_bool
1642   {
1643     \@@_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).

```

1644     \clist_if_empty:NF \l_@@_hlines_clist
1645     {
1646       \str_if_eq:eeF \l_@@_hlines_clist { all }
1647       {
1648         \clist_if_in:NeT
1649         \l_@@_hlines_clist
1650         { \int_eval:n { \c@iRow + 1 } }
1651       }
1652     }

```

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.

```

1653     \int_compare:nNnT \c@iRow > { -1 }
1654     {
1655       \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1656       { \hrule height \arrayrulewidth width \c_zero_dim }
1657     }
1658   }
1659 }
1660 }
1661 }

```

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

```

1662 \cs_set_protected:Npn \@@_renew_dots:
1663 {
1664   \cs_set_eq:NN \ldots \@@_Ldots:
1665   \cs_set_eq:NN \cdots \@@_Cdots:
1666   \cs_set_eq:NN \vdots \@@_Vdots:
1667   \cs_set_eq:NN \ddots \@@_Ddots:
1668   \cs_set_eq:NN \iddots \@@_Iddots:
1669   \cs_set_eq:NN \dots \@@_Ldots:
1670   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1671 }

```

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 ⁵.

```

1672 \AtBeginDocument
1673 {
1674   \IfPackageLoadedTF { booktabs }
1675   {
1676     \cs_new_protected:Npn \@@_patch_booktabs:
1677     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1678   }
1679   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1680 }

```

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

```

1681 \cs_new_protected:Npn \@@_some_initialization:
1682 {
1683   \@@_everycr:
1684   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1685   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1686   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1687   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1688   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1689   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1690 }

```

`\@@_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`.

```

1691 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1692 {
1693   % \tag_if_active:T % added on July 7 2026
1694   % {
1695   %   \AddToHookWithArguments{tbl/cell/init}
1696   %   {

```

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

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

```

1697 %          \int_show:n { ##2 }
1698 %          \int_show:N \g_@@_static_num_of_col_int
1699 %          \int_compare:nNnT { ##2 } > \g_@@_static_num_of_col_int
1700 %          { \SuspendTagging }
1701 %      }
1702 %  }

```

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

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

```

1703 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1704 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1705 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1706 \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.

```

1707 \fp_gzero:N \g_@@_total_X_weight_fp
1708 \bool_gset_false:N \g_@@_V_of_X_bool

1709 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1710 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1711 \@@_patch_booktabs:
1712 \box_clear_new:N \l_@@_cell_box
1713 \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}`).

```

1714 \bool_if:NT \l_@@_small_bool
1715 {
1716     \def \arraystretch { 0.47 }
1717     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1718 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1719 }

```

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!

```

1720 \bool_if:NT \g_@@_create_cell_nodes_bool
1721 {
1722     \tl_put_right:Nn \@@_begin_of_row:
1723     {
1724         \pgfsys@markposition
1725         { \@@_env: - row - \int_use:N \c@iRow - base }
1726     }
1727     \socket_assign_plugin { nicematrix / create-cell-nodes } { active }
1728 }

```

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`.

```

1729 \def \ar@ialign
1730 {
1731   \tbl_init_cell_data_for_table:
1732   \@@_some_initialization:
1733   \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`.

```

1734   \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1735   \halign
1736 }

```

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).

```

1737 \cs_set_eq:NN \@@_old_ldots: \ldots
1738 \cs_set_eq:NN \@@_old_cdots: \cdots
1739 \cs_set_eq:NN \@@_old_vdots: \vdots
1740 \cs_set_eq:NN \@@_old_ddots: \ddots
1741 \cs_set_eq:NN \@@_old_iddots: \iddots
1742 \bool_if:NTF \l_@@_standard_cline_bool
1743 { \cs_set_eq:NN \cline \@@_standard_cline: }
1744 { \cs_set_eq:NN \cline \@@_cline: }
1745 \cs_set_eq:NN \Ldots \@@_Ldots:
1746 \cs_set_eq:NN \Cdots \@@_Cdots:
1747 \cs_set_eq:NN \Vdots \@@_Vdots:
1748 \cs_set_eq:NN \Ddots \@@_Ddots:
1749 \cs_set_eq:NN \Iddots \@@_Iddots:
1750 \cs_set_eq:NN \Hline \@@_Hline:
1751 \cs_set_eq:NN \Hspace \@@_Hspace:
1752 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1753 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1754 \cs_set_eq:NN \Block \@@_Block:
1755 \cs_set_eq:NN \rotate \@@_rotate:
1756 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1757 \cs_set_eq:NN \dotfill \@@_dotfill:
1758 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1759 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1760 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1761 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1762 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1763 \cs_set_eq:NN \TopRule \@@_TopRule
1764 \cs_set_eq:NN \MidRule \@@_MidRule
1765 \cs_set_eq:NN \BottomRule \@@_BottomRule
1766 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1767 \cs_set_eq:NN \Hbrace \@@_Hbrace
1768 \cs_set_eq:NN \Vbrace \@@_Vbrace
1769 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1770 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1771 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1772 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1773 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1774 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1775 \cs_set_eq:NN \FalseRow \@@_FalseRow
1776 \int_if_zero:nTF \l_@@_first_row_int
1777 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1778 {
1779   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int

```

```

1780      { \cs_gset_eq:NN \@_tuning_exterior_rows: \@_tuning_last_row: }
1781    }
1782    \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:`.

```

1783    \cs_set_eq:NN \multicolumn \@_multicolumn:nnn
1784    \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1785      { \cs_set_eq:NN \multicolumn \@_old_multicolumn: }
1786    \@_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!).

```

1787    \tl_if_exist:NT \l_@_note_in_caption_tl
1788      {
1789        \tl_if_empty:NF \l_@_note_in_caption_tl
1790          {
1791            \int_gset:Nn \g_@_notes_caption_int \l_@_note_in_caption_tl
1792            \int_gset:Nn \c@tabularnote \l_@_note_in_caption_tl
1793          }
1794      }

```

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).

```

1795    \seq_gclear:N \g_@_multicolumn_cells_seq
1796    \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`).

```

1797    \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).

```

1798    \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`).

```

1799    \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.

```

1800    \int_gzero:N \g_@_col_total_int
1801    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1802    \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.

```

1803    \tl_gclear_new:N \g_@_Cdots_lines_tl
1804    \tl_gclear_new:N \g_@_Ldots_lines_tl
1805    \tl_gclear_new:N \g_@_Vdots_lines_tl
1806    \tl_gclear_new:N \g_@_Ddots_lines_tl
1807    \tl_gclear_new:N \g_@_Iddots_lines_tl
1808    \tl_gclear_new:N \g_@_HVDotsfor_lines_tl

```

```

1809 \tl_gclear:N \g_nicematrix_code_before_tl
1810 \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]`).

```

1811 \dim_zero_new:N \l_@@_left_delim_dim
1812 \dim_zero_new:N \l_@@_right_delim_dim
1813 \bool_if:NTF \g_@@_delims_bool
1814 {

```

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

```

1815 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1816 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1817 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1818 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1819 }
1820 {
1821 \dim_gset:Nn \l_@@_left_delim_dim
1822 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1823 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1824 }
1825 }

```

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.

```

1826 \cs_new_protected:Npn \@@_pre_array:
1827 {
1828 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1829 \int_gzero_new:N \c@iRow
1830 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1831 \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).

```

1832 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1833 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1834 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1835 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1836 \bool_if:NT \g_@@_aux_found_bool
1837 {
1838 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1839 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1840 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1841 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1842 }

```

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).

```

1843 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1844 {
1845 \bool_set_true:N \l_@@_last_row_without_value_bool
1846 \bool_if:NT \g_@@_aux_found_bool
1847 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1848 }

```

```

1849 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1850 {
1851     \bool_if:NT \g_@@_aux_found_bool
1852     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1853 }

```

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”.

```

1854 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1855 {
1856     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1857     {
1858         \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1859         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1860         \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1861         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1862     }
1863 }

1864 \seq_gclear:N \g_@@_cols_vlism_seq
1865 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1866 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1867 \@@_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).

```

1868 \hbox_set:Nw \l_@@_the_array_box
1869 \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1870 \skip_horizontal:N \l_@@_extra_left_margin_dim
1871 \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).

```

1872 \m@th
1873 $ % $
1874 \bool_if:NTF \l_@@_light_syntax_bool
1875 { \use:c { @@-light-syntax } }
1876 { \use:c { @@-normal-syntax } }
1877 }

```

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

```

1878 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1879 {
1880     \tl_set:Nn \l_tmpa_tl { #1 }
1881     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1882     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1883     \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1884     \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...

```

1885 \@@_pre_array:
1886 }

```

9 The \CodeBefore

```

1887 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1888 \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).

```

1889 \cs_new_protected:Npn \@@_pre_code_before:
1890 {

```

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.

```

1891 \pgfsys@markposition { \@@_env: - position }
1892 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1893 \pgfpicture
1894 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1895 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1896 {
1897     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1898     \pgfcoordinate { \@@_env: - row - ##1 }
1899     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1900 }

```

Now, the recreation of the col nodes.

```

1901 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1902 {
1903     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1904     \pgfcoordinate { \@@_env: - col - ##1 }
1905     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1906 }

```

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

```

1907 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1908 \endpgfpicture

```

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

```

1909 \@@_create_diag_nodes:

```

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

```

1910 \@@_create_blocks_nodes:
1911 \IfPackageLoadedT { tikz }
1912 {
1913     \tikzset
1914     {
1915         every-picture / .style =
1916         { overlay , name~prefix = \@@_env: - }
1917     }
1918 }
1919 \cs_set_eq:NN \cellcolor \@@_cellcolor
1920 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1921 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1922 \cs_set_eq:NN \rowcolor \@@_rowcolor
1923 \cs_set_eq:NN \rowcolors \@@_rowcolors
1924 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1925 \cs_set_eq:NN \arraycolor \@@_arraycolor
1926 \cs_set_eq:NN \columncolor \@@_columncolor
1927 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1928 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1929 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1930 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1931 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1932 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1933 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1934 }

```

```

1935 \cs_new_protected:Npn \@@_exec_code_before:
1936 {

```

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...

```

1937 \clist_map_inline:Nn \l_@@_corners_cells_clist
1938 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1939 \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`.

```

1940 \@@_add_to_colors_seq:nn { { nocolor } } { }
1941 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1942 \group_begin:

```

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

```

1943 \if_mode_math:
1944 \@@_exec_code_before_i:
1945 \else:
1946 $ % $
1947 \@@_exec_code_before_i:
1948 $ % $
1949 \fi:
1950 \group_end:
1951 }

```

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`).

```

1952 \cs_new_protected:Npn \@@_exec_code_before_i:
1953 {
1954 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1955 { \@@_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`.

```

1956 \exp_last_unbraced:No \@@_CodeBefore_keys:
1957 \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.

```

1958 \@@_actually_color:
1959 \l_@@_code_before_tl
1960 \q_stop
1961 }

```

```

1962 \keys_define:nn { nicematrix / CodeBefore }
1963 {
1964 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1965 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1966 sub-matrix .value_required:n = true ,
1967 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1968 delimiters / color .value_required:n = true ,
1969 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1970 }

```

```

1971 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1972 {
1973   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1974   \@@_CodeBefore:w
1975 }

```

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.

```

1976 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1977 {
1978   \bool_if:NTF \g_@@_aux_found_bool
1979   {
1980     \@@_pre_code_before:
1981     \legacy_if:nF { measuring@ } { #1 }
1982   }

```

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).

```

1983   {
1984     \pgfsys@markposition { \@@_env: - position }
1985     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1986     \pgfpicture
1987     \pgf@relevantforpicturesizefalse
1988     \endpgfpicture

```

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

```

1989     \pgfpicture
1990     \pgfrememberpicturepositiononpagetrue
1991     \endpgfpicture

```

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

```

1992     \pgfpicture
1993     \pgf@relevantforpicturesizefalse
1994     \pgfrememberpicturepositiononpagetrue
1995     \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1996     \pgfpicture
1997     \pgf@relevantforpicturesizefalse
1998     \endpgfpicture
1999   }
2000 }

```

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.

```

2001 \cs_new_protected:Npn \@@_recreate_cell_nodes:
2002 {
2003   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2004   {
2005     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
2006     \pgfcoordinate { \@@_env: - row - ##1 - base }
2007     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2008     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2009     {
2010       \cs_if_exist:cT
2011       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
2012       {
2013         \pgfsys@getposition
2014         { \@@_env: - ##1 - #####1 - NW }
2015         \@@_node_position:
2016         \pgfsys@getposition

```

```

2017         { \@@_env: - ##1 - ####1 - SE }
2018         \@@_node_position_i:
2019         \@@_pgf_rect_node:nnn
2020         { \@@_env: - ##1 - ####1 }
2021         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2022         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2023     }
2024 }
2025 }
2026 \@@_create_extra_nodes:
2027 \@@_create_aliases_last:
2028 }

2029 \cs_new_protected:Npn \@@_create_aliases_last:
2030 {
2031     \int_step_inline:nn \c@iRow
2032     {
2033         \pgfnodealias
2034         { \@@_env: - ##1 - last }
2035         { \@@_env: - ##1 - \int_use:N \c@jCol }
2036     }
2037     \int_step_inline:nn \c@jCol
2038     {
2039         \pgfnodealias
2040         { \@@_env: - last - ##1 }
2041         { \@@_env: - \int_use:N \c@iRow - ##1 }
2042     }
2043     \pgfnodealias
2044     { \@@_env: - last - last }
2045     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2046 }

2047 \cs_new_protected:Npn \@@_create_blocks_nodes:
2048 {
2049     \pgfpicture
2050     \pgf@relevantforpicturesizefalse
2051     \pgfrememberpicturepositiononpagetrue
2052     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2053     { \@@_create_one_block_node:nnnnn ##1 }
2054     \endpgfpicture
2055 }

```

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

```

2056 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2057 {
2058     \tl_if_empty:nF { #5 }
2059     {
2060         \@@_qpoint:n { col - #2 }
2061         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2062         \@@_qpoint:n { #1 }
2063         \dim_set_eq:NN \l_tmpb_dim \pgf@y
2064         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2065         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2066         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2067         \@@_pgf_rect_node:nnnnn
2068         { \@@_env: - #5 }
2069         { \dim_use:N \l_tmpa_dim }
2070         { \dim_use:N \l_tmpb_dim }

```

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

```

2071         { \dim_use:N \l_@@_tmpc_dim }
2072         { \dim_use:N \pgf@y }
2073     }
2074 }

```

10 The environment `{NiceArrayWithDelims}`

```

2075 \NewDocumentEnvironment { NiceArrayWithDelims }
2076 { m m 0 { } m ! 0 { } t \CodeBefore }
2077 {
2078     \@@_provide_pgfsyspdfmark:
2079     \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 exposit to a matrix in a mathematical formula.

```

2080     \bgroup

2081     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2082     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2083     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2084     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2085     \int_gzero:N \g_@@_block_box_int
2086     \dim_gzero:N \g_@@_width_last_col_dim
2087     \dim_gzero:N \g_@@_width_first_col_dim
2088     \bool_gset_false:N \g_@@_row_of_col_done_bool
2089     \str_if_empty:NT \g_@@_name_env_str
2090     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2091     \bool_if:NTF \l_@@_tabular_bool
2092     \mode_leave_vertical:
2093     \@@_test_if_math_mode:
2094     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2095     \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁸. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

2096     \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.

```

2097     \cs_if_exist:NT \tikz@library@external@loaded
2098     {
2099         \tikzexternaldisable
2100         \cs_if_exist:NT \ifstandalone
2101         { \tikzset { external / optimize = false } }
2102     }

```

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

```

2103     \int_gincr:N \g_@@_env_int
2104     \bool_if:NF \l_@@_block_auto_columns_width_bool
2105     { \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.

```

2106     \seq_gclear:N \g_@@_blocks_seq
2107     \seq_gclear:N \g_@@_pos_of_blocks_seq

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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

```

2108 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2109 \seq_gclear:N \g_@@_pos_of_xdots_seq
2110 \tl_gclear_new:N \g_@@_code_before_tl
2111 \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.

```

2112 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2113 {
2114   \bool_gset_true:N \g_@@_aux_found_bool
2115   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2116 }
2117 { \bool_gset_false:N \g_@@_aux_found_bool }

```

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

```

2118 \tl_gclear:N \g_@@_aux_tl
2119 \tl_if_empty:NF \g_@@_code_before_tl
2120 {
2121   \bool_set_true:N \l_@@_code_before_bool
2122   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2123 }
2124 \tl_if_empty:NF \g_@@_pre_code_before_tl
2125 { \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`.

```

2126 \bool_if:NTF \g_@@_delims_bool
2127 { \keys_set:nn { nicematrix / pNiceArray } }
2128 { \keys_set:nn { nicematrix / NiceArray } }
2129 { #3 , #5 }

2130 \@@_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:`.

```

2131 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2132 }

```

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

```

2133 {
2134   \bool_if:NTF \l_@@_light_syntax_bool
2135   { \use:c { end @@-light-syntax } }
2136   { \use:c { end @@-normal-syntax } }
2137   $ % $
2138   \skip_horizontal:N \l_@@_right_margin_dim
2139   \skip_horizontal:N \l_@@_extra_right_margin_dim
2140   \hbox_set_end:
2141   \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.

```

2142 \bool_if:NT \l_@@_width_used_bool
2143 {
2144   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }

```

```

2145     { \@@_error_or_warning:n { width-without-X-columns } }
2146 }

```

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 .

```

2147 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2148   \@@_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).

```

2149 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2150 {
2151   \bool_if:NF \l_@@_last_row_without_value_bool
2152   {
2153     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2154     {
2155       \@@_error:n { Wrong-last-row }
2156       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2157     }
2158   }
2159 }

```

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”.⁹

```

2160 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2161 \bool_if:NTF \g_@@_last_col_found_bool
2162 { \int_gdecr:N \c@jCol }
2163 {
2164   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2165   { \@@_error:n { last-col-not-used } }
2166 }

```

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

```

2167 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2168 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2169 { \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. 96).

```

2170 \int_if_zero:nT \l_@@_first_col_int
2171 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2172 \bool_if:nTF { ! \g_@@_delims_bool }
2173 {
2174   \str_if_eq:eeTF c \l_@@_baseline_tl
2175   \@@_use_arraybox_with_notes_c:
2176   {
2177     \str_if_eq:eeTF b \l_@@_baseline_tl
2178     \@@_use_arraybox_with_notes_b:
2179     \@@_use_arraybox_with_notes:
2180   }
2181 }

```

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).

```

2182 {
2183   \int_if_zero:nTF \l_@@_first_row_int

```

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

```

2184     {
2185         \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2186         \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2187     }
2188     { \dim_zero:N \l_tmpa_dim }

```

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

```

2189     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2190     {
2191         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2192         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2193     }
2194     { \dim_zero:N \l_tmpb_dim }
2195     \hbox_set:Nn \l_tmpa_box
2196     {
2197         \m@th
2198         $ % $
2199         \@@_color:o \l_@@_delimiters_color_tl
2200         \exp_after:wN \left \g_@@_left_delim_tl
2201         \vcenter
2202         {

```

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.

```

2203         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2204         \hbox
2205         {
2206             \bool_if:NTF \l_@@_tabular_bool
2207             { \skip_horizontal:n { - \tabcolsep } }
2208             { \skip_horizontal:n { - \arraycolsep } }
2209             \@@_use_arraybox_with_notes_c:
2210             \bool_if:NTF \l_@@_tabular_bool
2211             { \skip_horizontal:n { - \tabcolsep } }
2212             { \skip_horizontal:n { - \arraycolsep } }
2213         }

```

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

```

2214         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2215     }
2216     \exp_after:wN \right \g_@@_right_delim_tl
2217     $ % $
2218 }

```

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.

```

2219     \bool_if:NTF \l_@@_delimiters_max_width_bool
2220     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2221     \@@_put_box_in_flow:
2222 }

```

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. 97).

```

2223     \bool_if:NT \g_@@_last_col_found_bool
2224     { \skip_horizontal:N \g_@@_width_last_col_dim }
2225     \bool_if:NT \l_@@_preamble_bool
2226     {
2227         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2228         { \@@_err_columns_not_used: }
2229     }
2230     \@@_after_array:

```

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

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

```
2231 \egroup
```

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

```
2232 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2233 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2234 \iow_now:Ne \@mainaux
2235 {
2236   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2237   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2238   { \exp_not:o \g_@@_aux_tl }
2239 }
2240 \iow_now:Nn \@mainaux { \ExplSyntaxOff }
```

```
2241 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2242 }
```

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

```
2243 \cs_new_protected:Npn \@@_err_columns_not_used:
2244 {
2245   \@@_warning:n { columns~not~used }
2246   \cs_gset:Npn \@@_err_columns_not_used: { }
2247 }
```

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 .

```
2248 \cs_new_protected:Npn \@@_compute_width_X:
2249 {
2250   \tl_gput_right:Ne \g_@@_aux_tl
2251   {
2252     \bool_set_true:N \l_@@_X_columns_aux_bool
2253     \dim_set:Nn \l_@@_X_columns_dim
2254     {
```

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`

```
2255     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2256     { \dim_use:N \l_@@_X_columns_dim }
2257     {
2258       \dim_compare:nNnTF
2259       {
2260         \dim_abs:n
2261         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2262       }
2263       <
2264       { 0.001 pt }
2265       { \dim_use:N \l_@@_X_columns_dim }
2266       {
2267         \dim_eval:n
2268         {
2269           \l_@@_X_columns_dim
2270           +
2271           \fp_to_dim:n
2272           {
2273             (
2274               \dim_eval:n
2275               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2276             )

```

```

2277         / \fp_use:N \g_@@_total_X_weight_fp
2278     }
2279 }
2280 }
2281 }
2282 }
2283 }
2284 }

```

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`.

```

2285 \cs_new_protected:Npn \@@_transform_preamble:
2286 {
2287     \@@_transform_preamble_i:
2288     \@@_transform_preamble_ii:
2289 }
2290 \cs_new_protected:Npn \@@_transform_preamble_i:
2291 {
2292     \int_gzero:N \c@jCol

```

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

```

2293     \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2294     \bool_gset_false:N \g_tmpb_bool

```

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

```

2295     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2296     \int_zero:N \l_tmpa_int
2297     \tl_gclear:N \g_@@_array_preamble_tl
2298     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2299     {
2300         \tl_gset:Nn \g_@@_array_preamble_tl
2301             { ! { \skip_horizontal:N \arrayrulewidth } }
2302     }
2303     {
2304         \clist_if_in:NnT \l_@@_vlines_clist 1
2305         {
2306             \tl_gset:Nn \g_@@_array_preamble_tl
2307                 { ! { \skip_horizontal:N \arrayrulewidth } }
2308         }
2309     }

```

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

```

2310     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2311     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2312     \@@_replace_columncolor:
2313 }

```

```

2314 \cs_new_protected:Npn \@@_transform_preamble_ii:
2315 {

```

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

```

2316 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2317 {
2318 \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2319 { \bool_gset_true:N \g_@@_delims_bool }
2320 }
2321 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2322 \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).

```

2323 \int_if_zero:nTF \l_@@_first_col_int
2324 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_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_left:Nn \g_@@_array_preamble_tl { @ { } } }
2334 }
2335 }
2336 }
2337 }
2338 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2339 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2340 {
2341 \bool_if:NF \g_@@_delims_bool
2342 {
2343 \bool_if:NF \l_@@_tabular_bool
2344 {
2345 \clist_if_empty:NT \l_@@_vlines_clist
2346 {
2347 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2348 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2349 }
2350 }
2351 }
2352 }

```

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.

```

2353 \tag_if_active:F
2354 {

```

Moreover, when `{NiceTabular*}` is used, the mechanism can’t be used for technical reasons. We test that situation with `\l_@@_tabular_width_dim`.

```

2355 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2356 {
2357 \tl_gput_right:Nn \g_@@_array_preamble_tl
2358 { > { \@@_err_too_many_cols: } 1 }
2359 }
2360 }
2361 }

```

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.

```
2362 \cs_new_protected:Npn \@@_rec_preamble:n #1
2363 {
```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism `\csname...\endcsname`. Be careful: all these functions take in as first argument the letter (or token) itself.¹¹

```
2364 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2365 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2366 {
```

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

```
2367 \cs_if_exist:cTF { NC @ find @ #1 }
2368 {
2369 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2370 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2371 }
2372 {
2373 \str_if_eq:nnTF { #1 } { S }
2374 { \@@_fatal:n { unknown~column~type~S } }
2375 { \@@_fatal:nn { unknown~column~type } { #1 } }
2376 }
2377 }
2378 }
```

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

```
2379 \cs_new_protected:Npn \@@_c: #1
2380 {
2381 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2382 \tl_gclear:N \g_@@_pre_cell_tl
2383 \tl_gput_right:Nn \g_@@_array_preamble_tl
2384 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2385 \int_gincr:N \c@jCol
2386 \@@_rec_preamble_after_col:n
2387 }

2388 \cs_new_protected:Npn \@@_l: #1
2389 {
2390 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2391 \tl_gclear:N \g_@@_pre_cell_tl
2392 \tl_gput_right:Nn \g_@@_array_preamble_tl
2393 {
2394 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2395 l
2396 < \@@_cell_end:
2397 }
2398 \int_gincr:N \c@jCol
2399 \@@_rec_preamble_after_col:n
2400 }
```

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

```

2401 \cs_new_protected:Npn \@@_r: #1
2402 {
2403   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2404   \tl_gclear:N \g_@@_pre_cell_tl
2405   \tl_gput_right:Nn \g_@@_array_preamble_tl
2406   {
2407     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2408     r
2409     < \@@_cell_end:
2410   }
2411   \int_gincr:N \c@jCol
2412   \@@_rec_preamble_after_col:n
2413 }

```

For ! and @

```

2414 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2415 {
2416   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2417   \@@_rec_preamble:n
2418 }
2419 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2420 \cs_new_protected:cpn { @@ _ | : } #1
2421 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2422   \int_incr:N \l_tmpa_int
2423   \@@_make_preamble_i_i:n
2424 }

```

```

2425 \cs_new_protected:cpn { @@ _ F : } #1 % added 2026/06/28
2426 {
2427   \tl_gput_right:Nn \g_@@_array_preamble_tl
2428   {
2429     > {
2430       \skip_horizontal:n { -2 \col@sep + \l_@@_width_of_false_dim }
2431       \skip_horizontal:N \c_zero_dim
2432       \@@_cell_begin:
2433     }
2434     c
2435     < {
2436       \@@_cell_end:
2437       \columncolor { nocolor }
2438     }
2439   }
2440   \int_gincr:N \c@jCol
2441   \tl_gput_left:Ne \g_nicematrix_code_before_tl
2442   { \exp_not:N \EmptyColumn { \arabic{jCol} } }
2443   \@@_rec_preamble_after_col:n
2444 }
2445 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2446 {

```

Here, we can't use \str_if_eq:eeTF.

```

2447   \str_if_eq:nnTF { #1 } { | }
2448   { \use:c { @@ _ | : } | }
2449   { \@@_make_preamble_i_ii:nn { } #1 }
2450 }

```

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

```

2451 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2452 {

```

```

2453 \str_if_eq:nnTF { #2 } { [ ]
2454   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2455     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2456   }
2457 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2458   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2459 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2460   {
2461     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2462     \tl_gput_right:Ne \g_@@_array_preamble_tl
2463     {

```

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

```

2464     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2465   }
2466   \tl_gput_right:Ne \g_@@_rules_tl
2467   {

```

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:

```

2468   {
2469     \int_compare:nNnT \l_tmpa_int > 1
2470     { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2471     \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2472     \dim_set:Nn \l_@@_rule_width_after_dim
2473     { \dim_use:N \l_@@_rule_width_before_dim }
2474     \@@_draw_vrule:n { #2 }
2475   }

```

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.

```

2476   }
2477   \int_zero:N \l_tmpa_int
2478   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2479   \@@_rec_preamble:n #1
2480 }

```

```

2481 \cs_new_protected:cpn { @@ _ > : } #1 #2
2482   {
2483     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2484     \@@_rec_preamble:n
2485   }
2486 \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.

```

2487 \keys_define:nn { nicematrix / p-column }
2488   {
2489     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2490     r .value_forbidden:n = true ,
2491     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2492     c .value_forbidden:n = true ,

```

```

2493   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2494   l .value_forbidden:n = true ,
2495   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2496   S .value_forbidden:n = true ,
2497   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2498   p .value_forbidden:n = true ,
2499   t .meta:n = p ,
2500   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2501   m .value_forbidden:n = true ,
2502   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2503   b .value_forbidden:n = true
2504 }

```

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

```

2505 \cs_new_protected:Npn \@@_p: #1
2506 {
2507   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2508   \@@_make_preamble_ii_i:n
2509 }
2510 \cs_new_eq:NN \@@_b: \@@_p:
2511 \cs_new_eq:NN \@@_m: \@@_p:
2512 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2513 {
2514   \str_if_eq:nnTF { #1 } { [ ]
2515     { \@@_make_preamble_ii_ii:w [ ]
2516       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2517     }
2518   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2519     { \@@_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.

```

2520 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2521 {

```

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).

```

2522   \str_set:Nn \l_@@_hpos_col_str { j }
2523   \@@_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`.

```

2524   \setlength { \l_tmpa_dim } { #2 }
2525   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2526 }
2527 \cs_new_protected:Npn \@@_keys_p_column:n #1
2528 { \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.

```

2529 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2530 {

```

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

```

2531 \use:e
2532 {
2533   \@@_make_preamble_ii_vi:nnnnnnnn
2534   { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2535   { #1 }
2536   {
2537     \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.

```

2538 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2539 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2540 {

```

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

```

2541 \def \exp_not:N \l_@@_hpos_cell_tl
2542 { \str_lowercase:f { \l_@@_hpos_col_str } }
2543 }
2544 \IfPackageLoadedTF { ragged2e }
2545 {
2546   \str_case:on \l_@@_hpos_col_str
2547   {

```

The following `\exp_not:N` are mandatory.

```

2548 c { \exp_not:N \Centering }
2549 l { \exp_not:N \RaggedRight }
2550 r { \exp_not:N \RaggedLeft }
2551 }
2552 }
2553 {
2554   \str_case:on \l_@@_hpos_col_str
2555   {
2556     c { \exp_not:N \centering }
2557     l { \exp_not:N \raggedright }
2558     r { \exp_not:N \raggedleft }
2559   }
2560 }
2561 #3
2562 }
2563 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2564 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2565 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2566 { #2 }
2567 {
2568   \str_case:onF \l_@@_hpos_col_str
2569   {
2570     { j } { c }
2571     { si } { c }
2572   }

```

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

```

2573 { \str_lowercase:f \l_@@_hpos_col_str }
2574 }
2575 }

```

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

```

2576 \int_gincr:N \c@jCol
2577 \@@_rec_preamble_after_col:n
2578 }

```

#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*.

```

2579 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2580 {
2581   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2582   {
2583     \tl_gput_right:Nn \g_@@_array_preamble_tl
2584     { > \@@_test_if_empty_for_S: }
2585   }
2586   {
2587     \str_if_eq:eeTF { #7 } { varwidth }
2588     {
2589       \tl_gput_right:Nn \g_@@_array_preamble_tl
2590       { > \@@_test_if_empty_varwidth: }
2591     }
2592     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2593   }
2594   \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2595   \tl_gclear:N \g_@@_pre_cell_tl
2596   \tl_gput_right:Nn \g_@@_array_preamble_tl
2597   {
2598     > {

```

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.

```

2599       \dim_set:Nn \l_@@_col_width_dim { #2 }
2600       \@@_cell_begin:

```

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

```

2601       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2602       \everypar
2603       {
2604         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2605         \everypar { }
2606       }

```

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

```

2607       #3

```

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

```

2608       \g_@@_row_style_tl
2609       \arraybackslash
2610       #5
2611     }
2612     #8
2613     < {
2614       #6

```

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

```
2615         \finalstrut \@arstrutbox
2616         \use:c { end #7 }
```

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

```
2617         #4
2618         \@@_cell_end:
2619     }
2620 }
2621 }
```

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

```
2622 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2623 {
```

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 `&`).

```
2624     \group_align_safe_begin:
2625     \peek_meaning:NTF &
2626     \@@_the_cell_is_empty:
2627     {
2628         \peek_meaning:NTF \\\
2629         \@@_the_cell_is_empty:
2630         {
2631             \peek_meaning:NTF \crcr
2632             \@@_the_cell_is_empty:
2633             \group_align_safe_end:
2634         }
2635     }
2636 }
```

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

```
2637 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2638 {
2639     \group_align_safe_begin:
2640     \peek_meaning:NTF &
2641     \@@_the_cell_is_empty_varwidth:
2642     {
2643         \peek_meaning:NTF \\\
2644         \@@_the_cell_is_empty_varwidth:
2645         {
2646             \peek_meaning:NTF \crcr
2647             \@@_the_cell_is_empty_varwidth:
2648             \group_align_safe_end:
2649         }
2650     }
2651 }
2652 \cs_new_protected:Npn \@@_the_cell_is_empty:
2653 {
2654     \group_align_safe_end:
2655     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2656     {
```

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

```
2657         \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`).

```
2658         \skip_horizontal:N \l_@@_col_width_dim
2659     }
2660 }
```

```

2661 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2662 {
2663   \group_align_safe_end:
2664   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2665     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2666 }
2667 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2668 {
2669   \peek_meaning:NT \__siunitx_table_skip:n
2670     { \bool_gset_true:N \g_@@_empty_cell_bool }
2671 }

```

The following command will be used in `m-columns` in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more than the height of `\strutbox`, there is only one row.

```

2672 \cs_new_protected:Npn \@@_center_cell_box:
2673 {

```

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

```

2674   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2675     {
2676       \dim_compare:nNnT
2677         { \box_ht:N \l_@@_cell_box }
2678         >

```

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).

```

2679         { \box_ht:N \strutbox }
2680         {
2681           \hbox_set:Nn \l_@@_cell_box
2682             {
2683               \box_move_down:nn
2684                 {
2685                   ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2686                     + \baselineskip ) / 2
2687                 }
2688                 { \box_use:N \l_@@_cell_box }
2689             }
2690         }
2691     }
2692 }

```

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

```

2693 \cs_new_protected:Npn \@@_V: #1 #2
2694 {
2695   \str_if_eq:nnTF { #2 } { [ ] }
2696     { \@@_make_preamble_V_i:w [ ] }
2697     { \@@_make_preamble_V_i:w [ ] { #2 } }
2698 }
2699 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2700 { \@@_make_preamble_V_ii:nn { #1 } }
2701 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2702 {
2703   \str_set:Nn \l_@@_vpos_col_str { p }
2704   \str_set:Nn \l_@@_hpos_col_str { j }
2705   \@@_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`.

```

2706 \setlength { \l_tmpa_dim } { #2 }
2707 \IfPackageLoadedTF { varwidth }
2708 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2709 {
2710 \@@_error_or_warning:n { varwidth-not-loaded }
2711 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2712 }
2713 }
2714 % \end{macrocode}
2715 %
2716 % \medskip
2717 % For |w| and |W|
2718 % \begin{macrocode}
2719 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2720 \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.

```

2721 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2722 {
2723 \tl_if_in:nnTF { clrs } { #3 }
2724 { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2725 {
2726 \@@_error:nn { Invalid-argument-for-w } { #3 }
2727 \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2728 }
2729 }
2730 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2731 {
2732 \str_if_eq:nnTF { #3 } { s }
2733 { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2734 { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2735 }

```

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.

```

2736 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2737 {
2738 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2739 \tl_gclear:N \g_@@_pre_cell_tl
2740 \tl_gput_right:Nn \g_@@_array_preamble_tl
2741 {
2742 > {

```

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`.

```

2743 \setlength { \l_@@_col_width_dim } { #2 }
2744 \@@_cell_begin:
2745 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2746 }
2747 c
2748 < {
2749 \@@_cell_end_for_w_s:
2750 #1

```

```

2751         \@@_adjust_size_box:
2752         \box_use_drop:N \l_@@_cell_box
2753     }
2754 }
2755 \int_gincr:N \c@jCol
2756 \@@_rec_preamble_after_col:n
2757 }

```

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

```

2758 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2759 {
2760     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2761     \tl_gclear:N \g_@@_pre_cell_tl
2762     \tl_gput_right:Nn \g_@@_array_preamble_tl
2763     {
2764         > {

```

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`.

```

2765         \setlength { \l_@@_col_width_dim } { #4 }
2766         \hbox_set:Nw \l_@@_cell_box
2767         \@@_cell_begin:
2768         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2769     }
2770     c
2771     < {
2772         \@@_cell_end:
2773         \hbox_set_end:
2774         #1
2775         \@@_adjust_size_box:
2776         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2777     }
2778 }

```

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

```

2779     \int_gincr:N \c@jCol
2780     \@@_rec_preamble_after_col:n
2781 }

```

```

2782 \cs_new_protected:Npn \@@_special_W:
2783 {
2784     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2785     { \@@_warning:n { W-warning } }
2786 }

```

For S (of siunitx).

```

2787 \AtBeginDocument
2788 {
2789     \IfPackageLoadedT { siunitx }
2790     {
2791         \cs_new_protected:Npn \@@_S: #1 #2
2792         {
2793             \str_if_eq:nnTF { #2 } { [ ] }
2794             { \@@_make_preamble_S:w [ ] }
2795             { \@@_make_preamble_S:w [ ] { #2 } }
2796         }
2797     }
2798 }

```

```

2799 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2800 { \@@_make_preamble_S_i:n { #1 } }

2801 \cs_new_protected:Npn \@@_test_siunitx:
2802 {
2803   \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2804   { \@@_fatal:n { siunitx~too-old } }
2805   \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2806 }
2807 % \end{maacrocode}
2808 %
2809 % \begin{macrocode}
2810 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2811 {
2812   \@@_test_siunitx:
2813   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2814   \tl_gclear:N \g_@@_pre_cell_tl
2815   \tl_gput_right:Nn \g_@@_array_preamble_tl
2816   {
2817     > {

```

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).

```

2818       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2819       \keys_set:nn { siunitx } { #1 }
2820       \@@_cell_begin:
2821       \siunitx_cell_begin:w
2822     }
2823     c
2824     <
2825     {
2826       \siunitx_cell_end:

```

We want the value of `\l__siunitx_table_text_bool` available *after* `\@@_cell_end:` because we need it to know how to align our box after the construction of the PGF/TikZ node. That's why we use `\g_@@_cell_after_hook_tl` to reset the correct value of `\l__siunitx_table_text_bool` (of course, if will stay local within the cell of the underlying `\halign`).

```

2827       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2828       {
2829         \bool_if:NTF \l__siunitx_table_text_bool
2830         \bool_set_true:N
2831         \bool_set_false:N
2832         \l__siunitx_table_text_bool
2833       }
2834       \@@_cell_end:
2835     }
2836   }

```

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

```

2837       \int_gincr:N \c@jCol
2838       \@@_rec_preamble_after_col:n
2839     }

```

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

```

2840 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2841 {
2842   \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.

```

2843   \int_if_zero:nTF \c@jCol
2844   {
2845     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2846     {

```

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

```

2847         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2848         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2849         \@@_rec_preamble:n #2
2850     }
2851     {
2852         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2853         \@@_make_preamble_iv:nn { #1 } { #2 }
2854     }
2855 }
2856 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2857 }
2858 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2859 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2860 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2861 {
2862     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2863     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2864     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2865     {
2866         \@@_error:nn { delimiter~after~opening } { #2 }
2867         \@@_rec_preamble:n
2868     }
2869     { \@@_rec_preamble:n #2 }
2870 }

```

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

```

2871 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2872 { \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}`).

```

2873 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2874 {
2875     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2876     \tl_if_in:nnTF { ) ] \} } { #2 }
2877     { \@@_make_preamble_v:nnn #1 #2 }
2878     {
2879         \str_if_eq:nnTF \s_stop { #2 }
2880         {
2881             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2882             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2883             {
2884                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2885                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2886                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2887                 \@@_rec_preamble:n #2
2888             }
2889         }
2890         {
2891             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2892             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2893             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2894             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2895             \@@_rec_preamble:n #2
2896         }
2897     }
2898 }
2899 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2900 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }

```

```

2901 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2902 {
2903   \str_if_eq:nnTF \s_stop { #3 }
2904   {
2905     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2906     {
2907       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2908       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2909       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2910       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2911     }
2912     {
2913       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2914       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2915       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2916       \@@_error:nn { double~closing~delimiter } { #2 }
2917     }
2918   }
2919   {
2920     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2921     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2922     \@@_error:nn { double~closing~delimiter } { #2 }
2923     \@@_rec_preamble:n #3
2924   }
2925 }

2926 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2927 { \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 @{...}.

```

2928 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2929 {
2930   \str_if_eq:nnTF { #1 } { < }
2931   { \@@_rec_preamble_after_col_i:n }
2932   {
2933     \str_if_eq:nnTF { #1 } { @ }
2934     { \@@_rec_preamble_after_col_ii:n }
2935     {
2936       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2937       {
2938         \tl_gput_right:Nn \g_@@_array_preamble_tl
2939         { ! { \skip_horizontal:N \arrayrulewidth } }
2940       }
2941       {
2942         \clist_if_in:NeT \l_@@_vlines_clist
2943         { \int_eval:n { \c@jCol + 1 } }
2944         {
2945           \tl_gput_right:Nn \g_@@_array_preamble_tl
2946           { ! { \skip_horizontal:N \arrayrulewidth } }
2947         }
2948       }
2949       \@@_rec_preamble:n { #1 }
2950     }
2951   }
2952 }

2953 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2954 {
2955   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2956   \@@_rec_preamble_after_col:n
2957 }

```

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.

```

2958 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2959 {
2960   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2961   {
2962     \tl_gput_right:Nn \g_@@_array_preamble_tl
2963     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2964   }
2965   {
2966     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2967     {
2968       \tl_gput_right:Nn \g_@@_array_preamble_tl
2969       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2970     }
2971     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2972   }
2973   \@@_rec_preamble:n
2974 }

2975 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2976 {
2977   \tl_clear:N \l_tmpa_tl
2978   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2979   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2980 }

```

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

```

2981 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2982 { \@@_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.

```

2983 \cs_new_protected:Npn \@@_X: #1 #2
2984 {
2985   \str_if_eq:nnTF { #2 } { [ ]
2986     { \@@_make_preamble_X:w [ ] }
2987     { \@@_make_preamble_X:w [ ] #2 }
2988   }
2989   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2990   { \@@_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`.

```

2991 \keys_define:nn { nicematrix / X-column }
2992 {
2993   V .code:n =
2994   \IfPackageLoadedTF { varwidth }
2995   {
2996     \bool_set_true:N \l_@@_V_of_X_bool
2997     \bool_gset_true:N \g_@@_V_of_X_bool
2998   }
2999   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
3000   unknown .code:n =
3001   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
3002   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }

```

```

3003     { \@@_error_or_warning:n { invalid-weight } }
3004 }

```

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

```

3005 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
3006 {

```

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).

```

3007     \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).

```

3008     \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.

```

3009     \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

3010     \@@_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.

```

3011     \bool_set_false:N \l_@@_V_of_X_bool
3012     \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

3013     \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).

```

3014     \bool_if:NTF \l_@@_X_columns_aux_bool
3015     {
3016         \@@_make_preamble_ii_iv:nnn

```

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

```

3017         { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
3018         { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
3019         { \@@_no_update_width: }
3020     }

```

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.

```

3021     {
3022         \tl_gput_right:Nn \g_@@_array_preamble_tl
3023         {
3024             > {
3025                 \@@_cell_begin:
3026                 \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.

```

3027         \NotEmpty

```

The following code will nullify the box of the cell.

```

3028         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3029         { \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`.

```

3030         \begin { minipage } { 5 cm } \arraybackslash
3031     }
3032     c
3033     < {
3034         \end { minipage }
3035         \@@_cell_end:
3036     }
3037 }
3038 \int_gincr:N \c@jCol
3039 \@@_rec_preamble_after_col:n
3040 }
3041 }

3042 \cs_new_protected:Npn \@@_no_update_width:
3043 {
3044     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3045     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3046 }

```

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

```

3047 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3048 {
3049     \seq_gput_right:Ne \g_@@_cols_vlism_seq
3050     { \int_eval:n { \c@jCol + 1 } }
3051     \tl_gput_right:Ne \g_@@_array_preamble_tl
3052     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3053     \@@_rec_preamble:n
3054 }

```

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.

```

3055 \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).

```

3056 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3057 { \@@_fatal:n { Preamble-forgotten } }
3058 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3059 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3060 { @@ _ \token_to_str:N \hline : }
3061 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3062 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3063 { @@ _ \token_to_str:N \hline : }
3064 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3065 { @@ _ \token_to_str:N \hline : }
3066 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3067 { @@ _ \token_to_str:N \hline : }
3068 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3069 { @@ _ \token_to_str:N \hline : }
3070 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3071 { \@@_fatal:n { NiceTabularX~probably~required } }
3072 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3073 { @@ _ \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).

```
3074 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3075 {
```

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`.

```
3076 \multispan { #1 }
3077 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3078 \begingroup
3079 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
3080 \tl_gclear:N \g_@@_preamble_tl
3081 \@@_make_m_preamble:n #2 \q_stop
```

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

```
3082 \def \@addamp
3083 {
3084   \legacy_if:nTF { @firstamp }
3085   { \legacy_if_set_false:n { @firstamp } }
3086   { \@preamerr 5 }
3087 }
3088 \exp_args:No \@mkpream \g_@@_preamble_tl
3089 \@addtopreamble \@empty
3090 \endgroup
3091 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
3092 \int_compare:nNnT { #1 } > 1
3093 {
3094   \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3095   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3096   \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3097   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3098   {
3099     {
3100       \int_if_zero:nTF \c@jCol
3101       { \int_eval:n { \c@iRow + 1 } }
3102       { \int_use:N \c@iRow }
3103     }
3104     { \int_eval:n { \c@jCol + 1 } }
3105     {
3106       \int_if_zero:nTF \c@jCol
3107       { \int_eval:n { \c@iRow + 1 } }
3108       { \int_use:N \c@iRow }
3109     }
3110     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
3111 { }
3112 }
3113 }
```

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

```
3114 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3115 {
```

```

3116     \tl_gput_right:Nc \g_@@_pre_code_before_tl
3117     {
3118         \@@_rectanglecolor [ ##1 ]
3119         { \exp_not:n { ##2 } }
3120         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3121         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3122     }
3123     \ignorespaces
3124 }

```

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

```

3125     \def \@sharp { #3 }
3126     \@arstrut
3127     \@preamble
3128     \null

```

We add some lines.

```

3129     \int_gadd:Nn \c@jCol { #1 - 1 }
3130     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3131     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3132     \ignorespaces
3133 }

```

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`.

```

3134 \
3135 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3136 {
3137     \str_case:nnF { #1 }
3138     {
3139         c { \@@_make_m_preamble_i:n #1 }
3140         l { \@@_make_m_preamble_i:n #1 }
3141         X { \@@_make_m_preamble_i:n l }
3142         r { \@@_make_m_preamble_i:n #1 }
3143         > { \@@_make_m_preamble_ii:nn #1 }
3144         ! { \@@_make_m_preamble_ii:nn #1 }
3145         @ { \@@_make_m_preamble_ii:nn #1 }
3146         | { \@@_make_m_preamble_iii:n #1 }
3147         p { \@@_make_m_preamble_iv:nnn t #1 }
3148         m { \@@_make_m_preamble_iv:nnn c #1 }
3149         b { \@@_make_m_preamble_iv:nnn b #1 }
3150         w { \@@_make_m_preamble_v:nnnn { } #1 }
3151         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3152     }
3153 }
3154 {
3155     \cs_if_exist:cTF { NC @ find @ #1 }
3156     {
3157         \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3158         \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3159     }
3160     {
3161         \str_if_eq:nnTF { #1 } { S }
3162         { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3163         { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3164     }
3165 }
3166 }

```

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

```

3167 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3168 {

```

```

3169 \tl_gput_right:Nn \g_@@_preamble_tl
3170 {
3171   > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3172   #1
3173   < \@@_cell_end:
3174 }

```

We test for the presence of a <.

```

3175 \@@_make_m_preamble_x:n
3176 }

```

For >, ! and @

```

3177 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3178 {
3179   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3180   \@@_make_m_preamble:n
3181 }

```

For |

```

3182 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3183 {
3184   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3185   \@@_make_m_preamble:n
3186 }

```

For p, m and b

```

3187 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3188 {
3189   \tl_gput_right:Nn \g_@@_preamble_tl
3190   {
3191     > {
3192       \@@_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`.

```

3193 \setlength { \l_tmpa_dim } { #3 }
3194 \begin { minipage } [ #1 ] { \l_tmpa_dim }
3195 \mode_leave_vertical:
3196 \arraybackslash
3197 \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3198 }
3199 c
3200 < {
3201   \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3202   \end { minipage }
3203   \@@_cell_end:
3204 }
3205 }

```

We test for the presence of a <.

```

3206 \@@_make_m_preamble_x:n
3207 }

```

For w and W

```

3208 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3209 {
3210   \tl_gput_right:Nn \g_@@_preamble_tl
3211   {
3212     > {
3213       \dim_set:Nn \l_@@_col_width_dim { #4 }
3214       \hbox_set:Nw \l_@@_cell_box
3215       \@@_cell_begin:
3216       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }

```

```

3217     }
3218     c
3219     < {
3220         \@@_cell_end:
3221         \hbox_set_end:
3222         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3223         #1
3224         \@@_adjust_size_box:
3225         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3226     }
3227 }

```

We test for the presence of a <.

```

3228     \@@_make_m_preamble_x:n
3229 }

```

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

```

3230 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3231 {
3232     \str_if_eq:nnTF { #1 } { < }
3233     \@@_make_m_preamble_ix:n
3234     { \@@_make_m_preamble:n { #1 } }
3235 }
3236 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3237 {
3238     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3239     \@@_make_m_preamble_x:n
3240 }

```

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`).

```

3241 \cs_new_protected:Npn \@@_put_box_in_flow:
3242 {
3243     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3244     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3245     \str_if_eq:eeTF \l_@@_baseline_tl { c }
3246     { \box_use_drop:N \l_tmpa_box }
3247     \@@_put_box_in_flow_i:
3248 }

```

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

```

3249 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3250 {
3251     \pgfpicture
3252     \@@_qpoint:n { row - 1 }
3253     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3254     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3255     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3256     \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).

```

3257     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3258     {
3259         \int_set:Nn \l_tmpa_int
3260         { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3261         \bool_lazy_or:nnT
3262         { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3263         { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }

```

```

3264         {
3265             \@@_error:n { bad-value-for-baseline-line }
3266             \int_set:Nn \l_tmpa_int 1
3267         }
3268     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3269 }
3270 {
3271     \str_if_eq:eeTF t \l_@@_baseline_tl
3272     { \int_set:Nn \l_tmpa_int 1 }
3273     {
3274         \str_if_eq:eeTF b \l_@@_baseline_tl
3275         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3276         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3277     }
3278     \bool_lazy_or:nnT
3279     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3280     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3281     {
3282         \@@_error:n { bad-value-for-baseline }
3283         \int_set:Nn \l_tmpa_int 1
3284     }
3285     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3286         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3287     }
3288     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3289     \endpgfpicture
3290     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3291 }

```

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).

```

3292 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3293 {

```

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.

```

3294     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3295     {
3296         \int_compare:nNnT \c@jCol > 1
3297         {
3298             \box_set_wd:Nn \l_@@_the_array_box
3299             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3300         }
3301     }

```

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

```

3302     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3303     \bool_if:NT \l_@@_caption_above_bool
3304     {
3305         \tl_if_empty:NF \l_@@_caption_tl
3306         {
3307             \bool_set_false:N \g_@@_caption_finished_bool
3308             \int_gzero:N \c@tabularnote
3309             \@@_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).

```

3310         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3311         {
3312             \tl_gput_right:Ne \g_@@_aux_tl
3313             {
3314                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3315                 { \int_use:N \g_@@_notes_caption_int }
3316             }
3317             \int_gzero:N \g_@@_notes_caption_int
3318         }
3319     }
3320 }

```

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

```

3321     \hbox
3322     {
3323         \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.

```

3324         \@@_create_extra_nodes:
3325         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3326     }

```

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).

```

3327     \bool_lazy_any:nT
3328     {
3329         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3330         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3331         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3332     }
3333     {
3334         \bool_if:NTF \l_@@_notes_no_print_bool
3335         { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3336         \@@_tabular_notes:
3337     }
3338     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3339     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3340     \end { minipage }
3341 }

```

```

3342 \cs_new_protected:Npn \@@_insert_caption:
3343 {
3344     \tl_if_empty:NF \l_@@_caption_tl
3345     {
3346         \cs_if_exist:NTF \@captive
3347         { \@@_insert_caption_i: }
3348         { \@@_error:n { caption~outside~float } }
3349     }
3350 }

```

```

3351 \cs_new_protected:Npn \@@_insert_caption_i:
3352 {
3353     \group_begin:

```

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

```
3354 \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.

```
3355 \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3356 \tl_if_empty:NTF \l_@@_short_caption_tl
3357 \caption
3358 { \caption [ \l_@@_short_caption_tl ] }
3359 { \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.

```
3360 \bool_if:NF \g_@@_caption_finished_bool
3361 {
3362   \bool_gset_true:N \g_@@_caption_finished_bool
3363   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3364   \int_gzero:N \c@tabularnote
3365 }
3366 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3367 \group_end:
3368 }

3369 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3370 {
3371   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3372   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3373 }

3374 \cs_set_protected:Npn \@@_tabular_notes_error:
3375 { \@@_error:n { Bad-use-of-NiceTabularNotes } }

3376 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:

3377 \cs_set_protected:Npn \@@_tabular_notes:
3378 {
3379   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3380   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3381   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3382   \skip_vertical:N 0.65ex
```

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

```
3383 \group_begin:
3384 \l_@@_notes_code_before_tl
3385 \tl_if_empty:NF \g_@@_tabularnote_tl
3386 {
3387   \g_@@_tabularnote_tl \par
3388   \tl_gclear:N \g_@@_tabularnote_tl
3389 }
```

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.

```
3390 \int_compare:nNnT \c@tabularnote > \c_zero_int
3391 {
3392   \bool_if:NTF \l_@@_notes_para_bool
3393   {
3394     \begin { tabularnotes* }
3395     \seq_map_inline:Nn \g_@@_notes_seq
3396       { \@@_one_tabularnote:nn ##1 }
3397     \strut
3398     \end { tabularnotes* }
```

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

```

3399         \par
3400     }
3401     {
3402         \tabularnotes
3403         \seq_map_inline:Nn \g_@@_notes_seq
3404         { \@@_one_tabularnote:nn ##1 }
3405         \strut
3406         \endtabularnotes
3407     }
3408 }
3409 \unskip
3410 \group_end:
3411 \bool_if:NT \l_@@_notes_bottomrule_bool
3412 {
3413     \IfPackageLoadedTF { booktabs }
3414     {

```

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

```

3415         \skip_vertical:N \aboverulesep

```

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

```

3416         { \CT@arc@ \hrule height \heavyrulewidth }
3417     }
3418     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3419 }
3420 \l_@@_notes_code_after_tl
3421 \seq_gclear:N \g_@@_notes_seq
3422 \seq_gclear:N \g_@@_notes_in_caption_seq
3423 \int_gzero:N \c@tabularnote
3424 }

```

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.

```

3425 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3426 {
3427     \tl_if_novalue:nTF { #1 }
3428     { \item }
3429     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3430 }

```

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`.

```

3431 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3432 {
3433     \pgfpicture
3434     \@@_qpoint:n { row - 1 }
3435     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3436     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3437     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3438     \endpgfpicture
3439     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3440     \int_if_zero:nT \l_@@_first_row_int
3441     {
3442         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3443         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3444     }
3445     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3446 }

```

Now, the general case.

```
3447 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3448 {
```

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

```
3449 \str_if_eq:eeT \l_@@_baseline_tl { t }
3450 { \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`.

```
3451 \pgfpicture
3452 \@@_qpoint:n { row - 1 }
3453 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3454 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3455 {
3456 \int_set:Nn \l_tmpa_int
3457 {
3458 \str_range:Nnn
3459 \l_@@_baseline_tl
3460 { 6 }
3461 { \tl_count:o \l_@@_baseline_tl }
3462 }
3463 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3464 }
3465 {
3466 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3467 \bool_lazy_or:nnT
3468 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3469 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3470 {
3471 \@@_error:n { bad~value~for~baseline }
3472 \int_set:Nn \l_tmpa_int 1
3473 }
3474 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3475 }
3476 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3477 \endpgfpicture
3478 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3479 \int_if_zero:nT \l_@@_first_row_int
3480 {
3481 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3482 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3483 }
3484 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3485 }
```

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.

```
3486 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3487 {
```

We will compute the real width of both delimiters used.

```
3488 \dim_zero_new:N \l_@@_real_left_delim_dim
3489 \dim_zero_new:N \l_@@_real_right_delim_dim
3490 \hbox_set:Nn \l_tmpb_box
3491 {
3492 \m@th
3493 $ % $
3494 \left #1
3495 \vcenter
3496 {
3497 \vbox_to_ht:nn
3498 { \box_ht_plus_dp:N \l_tmpa_box }
```

```

3499         { }
3500     }
3501     \right .
3502     $ % $
3503 }
3504 \dim_set:Nn \l_@@_real_left_delim_dim
3505 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3506 \hbox_set:Nn \l_tmpb_box
3507 {
3508     \m@th
3509     $ % $
3510     \left .
3511     \vbox_to_ht:nn
3512     { \box_ht_plus_dp:N \l_tmpa_box }
3513     { }
3514     \right #2
3515     $ % $
3516 }
3517 \dim_set:Nn \l_@@_real_right_delim_dim
3518 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3519     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3520     \@@_put_box_in_flow:
3521     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3522 }

```

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).

```

3523 \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).

```

3524 {
3525     \peek_remove_spaces:n
3526     {
3527         \peek_meaning:NTF \end
3528         \@@_analyze_end:Nn
3529         {
3530             \@@_transform_preamble:
3531             \@@_array:o \g_@@_array_preamble_tl
3532         }
3533     }
3534 }
3535 {
3536     \@@_create_col_nodes:
3537     \endarray
3538 }

```

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

```

3539 \NewDocumentEnvironment { @@-light-syntax } { b }
3540 {

```

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`.

```

3541     \tl_if_empty:nT { #1 }

```

```

3542     { \@@_fatal:n { empty-environment } }
3543 \tl_if_in:nnT { #1 } { & }
3544     { \@@_fatal:n { ampersand-in~light-syntax } }
3545 \tl_if_in:nnT { #1 } { \ }
3546     { \@@_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`.

```

3547 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3548 }

```

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.

```

3549 {
3550   \@@_create_col_nodes:
3551   \endarray
3552 }
3553 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3554 {
3555   \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).

```

3556 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3557 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3558 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3559   \seq_set_split:Nee
3560   \seq_set_split:Non
3561   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3562 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3563 \tl_if_empty:NF \l_tmpa_tl
3564   { \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.

```

3565 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3566   { \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`.

```

3567 \tl_build_begin:N \l_@@_new_body_tl
3568 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3569 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3570 \@@_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).

```

3571 \seq_map_inline:Nn \l_@@_rows_seq
3572 {
3573   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3574   \@@_line_with_light_syntax:n { ##1 }
3575 }
3576 \tl_build_end:N \l_@@_new_body_tl

```

```

3577 \int_compare:nNtT \l_@@_last_col_int = { -1 }
3578 {
3579     \int_set:Nn \l_@@_last_col_int
3580     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3581 }

```

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.

```

3582 \@@_transform_preamble:

3583 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3584 }

3585 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3586 {
3587     \seq_clear_new:N \l_@@_cells_seq
3588     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3589     \int_set:Nn \l_@@_nb_cols_int
3590     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3591     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3592     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3593     \seq_map_inline:Nn \l_@@_cells_seq
3594     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3595 }
3596 \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`.

```

3597 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3598 {
3599     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3600     { \@@_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.

```

3601 \end { #2 }
3602 }

```

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`).

```

3603 \cs_new:Npn \@@_create_col_nodes:
3604 {
3605     \crrc
3606     \int_if_zero:nT \l_@@_first_col_int
3607     {
3608         \omit
3609         \hbox_overlap_left:n
3610         {
3611             \bool_if:NT \l_@@_code_before_bool
3612             { \pgfsys@markposition { \@@_env: - col - 0 } }
3613             \pgfpicture
3614             \pgfrememberpicturepositiononpagetrue
3615             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3616             \str_if_empty:NF \l_@@_name_str
3617             { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3618             \endpgfpicture
3619             \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3620         }
3621         &
3622     }
3623     \omit

```

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

```
3624 \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`).

```
3625 \int_if_zero:nTF \l_@@_first_col_int
3626 {
3627   \@@_mark_position:n { 1 }
3628   \pgfpicture
3629   \pgfrememberpicturepositiononpagetrue
3630   \pgfcoordinate { \@@_env: - col - 1 }
3631   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3632   \str_if_empty:NF \l_@@_name_str
3633   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3634   \endpgfpicture
3635 }
3636 {
3637   \bool_if:NT \l_@@_code_before_bool
3638   {
3639     \hbox
3640     {
3641       \skip_horizontal:n { 0.5 \arrayrulewidth }
3642       \pgfsys@markposition { \@@_env: - col - 1 }
3643       \skip_horizontal:n { -0.5 \arrayrulewidth }
3644     }
3645   }
3646   \pgfpicture
3647   \pgfrememberpicturepositiononpagetrue
3648   \pgfcoordinate { \@@_env: - col - 1 }
3649   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3650   \@@_node_alias:n { 1 }
3651   \endpgfpicture
3652 }
```

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.

```
3653 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3654 \bool_if:NF \l_@@_auto_columns_width_bool
3655 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3656 {
3657   \bool_lazy_and:nnTF
3658     \l_@@_auto_columns_width_bool
3659     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3660     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3661     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3662   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3663 }
3664 \skip_horizontal:N \g_tmpa_skip
3665 \hbox
3666 {
3667   \@@_mark_position:n { 2 }
3668   \pgfpicture
3669   \pgfrememberpicturepositiononpagetrue
3670   \pgfcoordinate { \@@_env: - col - 2 }
3671   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3672   \@@_node_alias:n { 2 }
3673   \endpgfpicture
3674 }
```

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.

```

3675 \int_gset:Nn \g_tmpa_int 1
3676 \bool_if:NTF \g_@@_last_col_found_bool
3677 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3678 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3679 {
3680   &
3681   \omit
3682   \int_gincr:N \g_tmpa_int

```

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

```

3683 \skip_horizontal:N \g_tmpa_skip
3684 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3685 \pgfpicture
3686 \pgfrememberpicturepositiononpagetrue
3687 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3688 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3689 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3690 \endpgfpicture
3691 }

```

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).

```

3692 \bool_lazy_or:nnF
3693 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3694 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3695 {
3696   &
3697   \omit
3698   \skip_horizontal:N \g_tmpa_skip
3699   \int_gincr:N \g_tmpa_int
3700   \bool_lazy_any:nF
3701   {
3702     \g_@@_delims_bool
3703     \l_@@_tabular_bool
3704     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3705     \l_@@_exterior_arraycolsep_bool
3706     \l_@@_bar_at_end_of_pream_bool
3707   }
3708   { \skip_horizontal:n { - \col@sep } }
3709   \bool_if:NT \l_@@_code_before_bool
3710   {
3711     \hbox
3712     {
3713       \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.

```

3714 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3715 { \skip_horizontal:n { - \arraycolsep } }
3716 \pgfsys@markposition
3717 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3718 \skip_horizontal:n { 0.5 \arrayrulewidth }
3719 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3720 { \skip_horizontal:N \arraycolsep }
3721 }
3722 }
3723 \pgfpicture
3724 \pgfrememberpicturepositiononpagetrue
3725 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3726 {

```

```

3727         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3728         {
3729             \pgfpoint
3730             { - 0.5 \arrayrulewidth - \arraycolsep }
3731             \c_zero_dim
3732         }
3733         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3734     }
3735     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3736 \endpgfpicture
3737 }

3738 \bool_if:NT \g_@@_last_col_found_bool
3739 {
3740     \hbox_overlap_right:n
3741     {
3742         \skip_horizontal:N \g_@@_width_last_col_dim
3743         \skip_horizontal:N \colsep
3744         \bool_if:NT \l_@@_code_before_bool
3745         {
3746             \pgfsys@markposition
3747             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3748         }
3749         \pgfpicture
3750         \pgfrememberpicturepositiononpagetrue
3751         \pgfcoordinate
3752         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3753         \pgfpointorigin
3754         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3755         \endpgfpicture
3756     }
3757 }
3758 }

3759 \cs_new_protected:Npn \@@_mark_position:n #1
3760 {
3761     \bool_if:NT \l_@@_code_before_bool
3762     {
3763         \hbox
3764         {
3765             \skip_horizontal:n { -0.5 \arrayrulewidth }
3766             \pgfsys@markposition { \@@_env: - col - #1 }
3767             \skip_horizontal:n { 0.5 \arrayrulewidth }
3768         }
3769     }
3770 }

3771 \cs_new_protected:Npn \@@_node_alias:n #1
3772 {
3773     \str_if_empty:NF \l_@@_name_str
3774     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3775 }

```

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

```

3776 \tl_const:Nn \c_@@_preamble_first_col_tl
3777 {
3778     >
3779     {

```

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

```

3780 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3781 \bool_gset_true:N \g_@@_after_col_zero_bool
3782 \@@_begin_of_row:
3783 \hbox_set:Nw \l_@@_cell_box
3784 \@@_math_toggle:
3785 \@@_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”.

```

3786 \int_compare:nNnT \c_iRow > \c_zero_int
3787 {
3788   \bool_lazy_or:nnT
3789   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3790   { \int_compare_p:nNn \c_iRow < \l_@@_last_row_int }
3791   {
3792     \l_@@_code_for_first_col_tl
3793     \xglobal \colorlet { nicematrix-first-col } { . }
3794   }
3795 }
3796 }

```

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`.

```

3797   l
3798   <
3799   {
3800     \@@_math_toggle:
3801     \hbox_set_end:
3802     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3803     \@@_adjust_size_box:
3804     \@@_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.

```

3805 \dim_gset:Nn \g_@@_width_first_col_dim
3806 { \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.

```

3807 \hbox_overlap_left:n
3808 {
3809   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3810   \@@_node_cell:
3811   { \box_use_drop:N \l_@@_cell_box }
3812   \skip_horizontal:N \l_@@_left_delim_dim
3813   \skip_horizontal:N \l_@@_left_margin_dim
3814   \skip_horizontal:N \l_@@_extra_left_margin_dim
3815 }
3816 \bool_gset_false:N \g_@@_empty_cell_bool
3817 \skip_horizontal:n { -2 \col@sep }
3818 }
3819 }

```

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

```

3820 \tl_const:Nn \c_@@_preamble_last_col_tl
3821 {
3822   >
3823   {
3824     \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).

```

3825 \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.

```

3826 \bool_gset_true:N \g_@@_last_col_found_bool
3827 \int_gincr:N \c@jCol
3828 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3829 \hbox_set:Nw \l_@@_cell_box
3830 \@@_math_toggle:
3831 \@@_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”.

```

3832 \int_compare:nNnT \c@iRow > \c_zero_int
3833 {
3834   \bool_lazy_or:nnT
3835   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3836   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3837   {
3838     \l_@@_code_for_last_col_tl
3839     \xglobal \colorlet { nicematrix-last-col } { . }
3840   }
3841 }
3842 }
3843 1
3844 <
3845 {
3846   \@@_math_toggle:
3847   \hbox_set_end:
3848   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3849   \@@_adjust_size_box:
3850   \@@_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.

```

3851 \dim_gset:Nn \g_@@_width_last_col_dim
3852 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3853 \skip_horizontal:n { -2 \col@sep }

```

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

```

3854 \hbox_overlap_right:n
3855 {
3856   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3857   {
3858     \skip_horizontal:N \l_@@_right_delim_dim
3859     \skip_horizontal:N \l_@@_right_margin_dim
3860     \skip_horizontal:N \l_@@_extra_right_margin_dim
3861     \@@_node_cell:
3862   }
3863 }
3864 \bool_gset_false:N \g_@@_empty_cell_bool
3865 }
3866 }

```

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

```

3867 \NewDocumentEnvironment { NiceArray } { }
3868 {
3869   \bool_gset_false:N \g_@@_delims_bool
3870   \str_if_empty:NT \g_@@_name_env_str
3871   { \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).

```

3872 \NiceArrayWithDelims . .
3873 }
3874 { \endNiceArrayWithDelims }

```

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

```

3875 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3876 {
3877   \NewDocumentEnvironment { #1 NiceArray } { }
3878   {
3879     \bool_gset_true:N \g_@@_delims_bool
3880     \str_if_empty:NT \g_@@_name_env_str
3881     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3882     \@@_test_if_math_mode:
3883     \NiceArrayWithDelims #2 #3
3884   }
3885   { \endNiceArrayWithDelims }
3886 }
3887 \@@_def_env:NNN p ( )
3888 \@@_def_env:NNN b [ ]
3889 \@@_def_env:NNN B \{ \}
3890 \@@_def_env:NNN v \vert \vert
3891 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

13.1 Definition of `{pNiceMatrix}`

```

3892 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3893 {
3894   \bool_set_false:N \l_@@_preamble_bool
3895   \tl_clear:N \l_tmpa_tl
3896   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3897   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3898   \tl_put_right:Nn \l_tmpa_tl
3899   {
3900     *
3901     {
3902       \int_case:nnF \l_@@_last_col_int
3903       {
3904         { -2 } { \c@MaxMatrixCols }
3905         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

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

```

3906       }
3907       { \int_eval:n { \l_@@_last_col_int - 1 } }
3908     }
3909     { #2 }
3910   }
3911   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3912   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3913 }
3914 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3915 \clist_map_inline:nn { p , b , B , v , V }
3916 {
3917   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3918   {
3919     \bool_gset_true:N \g_@@_delims_bool
3920     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3921     \int_if_zero:nT \l_@@_last_col_int
3922     {
3923       \bool_set_true:N \l_@@_last_col_without_value_bool
3924       \int_set:Nn \l_@@_last_col_int { -1 }

```

```

3925     }
3926     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3927     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3928   }
3929   { \use:c { end #1 NiceArray } }
3930 }

```

We define also an environment {NiceMatrix}

```

3931 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3932 {
3933   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3934   \int_if_zero:nT \l_@@_last_col_int
3935   {
3936     \bool_set_true:N \l_@@_last_col_without_value_bool
3937     \int_set:Nn \l_@@_last_col_int { -1 }
3938   }
3939   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3940   \bool_lazy_or:nnT
3941     \l_@@_except_borders_bool
3942     { \clist_if_empty_p:N \l_@@_vlines_clist }
3943     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3944   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3945 }
3946 { \endNiceArray }

```

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

```

3947 \cs_new_protected:Npn \@@_NotEmpty:
3948 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

3949 \cs_set_protected:Npn \@@_renew_matrix:
3950 {
3951   \tl_map_inline:nn { pvVbB }
3952     { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3953 }

```

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

```

3954 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3955 {

```

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.

```

3956   \dim_compare:nNtT\l_@@_width_dim = \c_zero_dim
3957     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3958   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3959   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3960   \tl_if_empty:NF \l_@@_short_caption_tl
3961   {
3962     \tl_if_empty:NT \l_@@_caption_tl
3963     {
3964       \@@_error_or_warning:n { short-caption-without~caption }
3965       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3966     }
3967   }
3968   \tl_if_empty:NF \l_@@_label_tl
3969   {
3970     \tl_if_empty:NT \l_@@_caption_tl
3971       { \@@_error_or_warning:n { label~without~caption } }
3972   }

```

```

3973 \NewDocumentEnvironment { TabularNote } { b }
3974 {
3975     \bool_if:NTF \l_@@_in_code_after_bool
3976     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3977     {
3978         \tl_if_empty:NF \g_@@_tabularnote_tl
3979         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3980         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3981     }
3982 }
3983 { }
3984 \@@_settings_for_tabular:
3985 \NiceArray { #2 }
3986 }
3987 { \endNiceArray }
3988 \cs_new_protected:Npn \@@_settings_for_tabular:
3989 {
3990     \bool_set_true:N \l_@@_tabular_bool
3991     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3992     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3993     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3994 }

3995 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3996 {
3997     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3998     \dim_set:Nn \l_@@_width_dim { #1 }
3999     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4000     \@@_settings_for_tabular:
4001     \NiceArray { #3 }
4002 }
4003 {
4004     \endNiceArray
4005     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
4006     { \@@_error:n { NiceTabularX~without~X } }
4007 }

4008 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
4009 {
4010     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
4011     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
4012     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4013     \@@_settings_for_tabular:
4014     \NiceArray { #3 }
4015 }
4016 { \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`).

```

4017 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
4018 {
4019     \bool_lazy_all:nT
4020     {
4021         { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
4022         { \l_@@_hvlines_bool }
4023         { ! \g_@@_delims_bool }

```

```

4024     { ! \l_@@_except_borders_bool }
4025   }
4026   {
4027     \bool_set_true:N \l_@@_except_borders_bool
4028     \clist_if_empty:NF \l_@@_corners_clist
4029     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
4030     \tl_gput_right:Nn \g_@@_rules_tl
4031     {
4032       \@@_stroke_block:nnnnn
4033       {
4034         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
4035         draw = \l_@@_rules_color_tl
4036       }
4037       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4038     }
4039   }
4040 }

4041 \cs_new_protected:Npn \@@_after_array:
4042 {

```

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`.

```

4043   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
4044   \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.

```

4045   \bool_if:NT \g_@@_last_col_found_bool
4046   { \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`.

```

4047   \bool_if:NT \l_@@_last_col_without_value_bool
4048   { \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.

```

4049   \bool_if:NT \l_@@_last_row_without_value_bool
4050   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4051   \tl_gput_right:Ne \g_@@_aux_tl
4052   {
4053     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4054     {
4055       \int_use:N \l_@@_first_row_int ,
4056       \int_use:N \c@iRow ,
4057       \int_use:N \g_@@_row_total_int ,
4058       \int_use:N \l_@@_first_col_int ,
4059       \int_use:N \c@jCol ,
4060       \int_use:N \g_@@_col_total_int
4061     }
4062   }

4063   \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`).

```

4064 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4065 {
4066   \tl_gput_right:Ne \g_@@_aux_tl
4067   {
4068     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4069     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4070   }
4071 }
4072 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4073 {
4074   \tl_gput_right:Ne \g_@@_aux_tl
4075   {
4076     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4077     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4078     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4079     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4080   }
4081 }

```

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

```

4082 \@@_create_diag_nodes:

```

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

```

4083 \pgfpicture
4084 \@@_create_aliases_last:
4085 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4086 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

4087 \bool_if:NT \l_@@_parallelize_diags_bool
4088 {
4089   \int_gzero:N \g_@@_ddots_int
4090   \int_gzero:N \g_@@_iddots_int

```

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

```

4091 \dim_gzero:N \g_@@_delta_x_one_dim
4092 \dim_gzero:N \g_@@_delta_y_one_dim
4093 \dim_gzero:N \g_@@_delta_x_two_dim
4094 \dim_gzero:N \g_@@_delta_y_two_dim
4095 }
4096 \bool_set_false:N \l_@@_initial_open_bool
4097 \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.

```

4098 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4099 \@@_draw_dotted_lines:

```

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

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

4100   \clist_if_empty:NF \l_@@_corners_cells_clist
4101   {
4102     \bool_if:NTF \l_@@_no_cell_nodes_bool
4103     { \@@_error:n { corners-with-no-cell-nodes } }
4104     \@@_compute_corners:
4105   }

```

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

```

4106   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4107   \g_@@_pos_of_blocks_seq
4108   \g_@@_future_pos_of_blocks_seq
4109   \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-*}`).

```

4110   \@@_adjust_pos_of_blocks_seq:
4111   \@@_deal_with_rounded_corners:
4112   \legacy_if:nF { measuring@ } \@@_draw_rules:
4113   \tl_gclear:N \g_@@_rules_tl

```

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

```

4114   \IfPackageLoadedT { tikz }
4115   {
4116     \tikzset
4117     {
4118       every~picture / .style =
4119       {
4120         overlay ,
4121         remember~picture ,
4122         name~prefix = \@@_env: -
4123       }
4124     }
4125   }
4126   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4127   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4128   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4129   \cs_set_eq:NN \OverBrace \@@_OverBrace
4130   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4131   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4132   \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.

```

4133   \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4134   \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.

```

4135   \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).

```

4136   \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`).

```
4137 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4138 { \@@_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:`.

```
4139 \bool_set_true:N \l_@@_in_code_after_bool
4140 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4141 \scan_stop:
4142 \tl_gclear:N \g_nicematrix_code_after_tl
4143 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4144 \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.

```
4145 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4146 \tl_if_empty:NF \g_@@_pre_code_before_tl
4147 {
4148   \tl_gput_right:Ne \g_@@_aux_tl
4149   {
4150     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4151     { \exp_not:o \g_@@_pre_code_before_tl }
4152   }
4153   \tl_gclear:N \g_@@_pre_code_before_tl
4154 }
4155 \tl_if_empty:NF \g_nicematrix_code_before_tl
4156 {
4157   \tl_gput_right:Ne \g_@@_aux_tl
4158   {
4159     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4160     { \exp_not:o \g_nicematrix_code_before_tl }
4161   }
4162   \tl_gclear:N \g_nicematrix_code_before_tl
4163 }

4164 \str_gclear:N \g_@@_name_env_str
4165 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```
4166 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4167 }

4168 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4169 {
4170   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4171   \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.

```
4172 \dim_set:Nn \l_@@_xdots_shorten_start_dim
4173 { 0.6 \l_@@_xdots_shorten_start_dim }
4174 \dim_set:Nn \l_@@_xdots_shorten_end_dim
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

4175     { 0.6 \l_@@_xdots_shorten_end_dim }
4176 }

```

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`.

```

4177 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4178 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4179 \cs_new_protected:Npn \@@_create_alias_nodes:
4180 {
4181   \int_step_inline:nn \c@iRow
4182   {
4183     \pgfnodealias
4184     { \l_@@_name_str - ##1 - last }
4185     { \@@_env: - ##1 - \int_use:N \c@jCol }
4186   }
4187   \int_step_inline:nn \c@jCol
4188   {
4189     \pgfnodealias
4190     { \l_@@_name_str - last - ##1 }
4191     { \@@_env: - \int_use:N \c@iRow - ##1 }
4192   }
4193   \pgfnodealias
4194   { \l_@@_name_str - last - last }
4195   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4196 }

```

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).

```

4197 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4198 {
4199   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4200   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4201 }

```

The following command must *not* be protected.

```

4202 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4203 {
4204   { #1 }
4205   { #2 }
4206   {
4207     \int_compare:nNnTF { #3 } > { 98 }
4208     { \int_use:N \c@iRow }
4209     { #3 }
4210   }
4211   {
4212     \int_compare:nNnTF { #4 } > { 98 }
4213     { \int_use:N \c@jCol }
4214     { #4 }
4215   }
4216   { #5 }
4217 }

```

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).

```

4218 \AtBeginDocument
4219 {
4220   \cs_new_protected:Npn \@@_draw_dotted_lines:
4221   {
4222     \c_@@_pgfortikzpicture_tl
4223     \@@_draw_dotted_lines_i:
4224     \c_@@_endpgfortikzpicture_tl
4225   }
4226 }

```

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

```

4227 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4228 {
4229   \pgfrememberpicturepositiononpagetrue
4230   \pgf@relevantforpicturesizefalse
4231   \g_@@_HVdotsfor_lines_tl
4232   \g_@@_Vdots_lines_tl
4233   \g_@@_Ddots_lines_tl
4234   \g_@@_Iddots_lines_tl
4235   \g_@@_Cdots_lines_tl
4236   \g_@@_Ldots_lines_tl
4237 }

4238 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4239 {
4240   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4241   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4242 }

```

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

```

4243 \pgfdeclareshape { @@_diag_node }
4244 {
4245   \savedanchor { \five }
4246   {
4247     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4248     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4249   }
4250   \anchor { 5 } { \five }
4251   \anchor { center } { \pgfpointorigin }
4252   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4253   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4254   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4255   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4256   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4257   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4258   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4259   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4260   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4261   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4262 }

```

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).

```

4263 \cs_new_protected:Npn \@@_create_diag_nodes:
4264 {
4265   \pgfpicture

```

```

4266 \pgfrememberpicturepositiononpagetrue
4267 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4268 {
4269     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4270     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4271     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4272     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4273     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4274     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4275     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4276     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4277     \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.

```

4278     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4279     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4280     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4281     \str_if_empty:NF \l_@@_name_str
4282     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4283 }

```

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

```

4284 \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4285 \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4286 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4287 \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4288 \pgfcoordinate
4289 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4290 \pgfnodealias
4291 { \@@_env: - last }
4292 { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4293 \str_if_empty:NF \l_@@_name_str
4294 {
4295     \pgfnodealias
4296     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4297     { \@@_env: - \int_use:N \l_tmpa_int }
4298     \pgfnodealias
4299     { \l_@@_name_str - last }
4300     { \@@_env: - last }
4301 }
4302 \endpgfpicture
4303 }

```

16 We draw the dotted lines

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

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

The command `\@@_find_extremities:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;

- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \l_@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {
        \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
  {
    \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
    {
      \int_compare:nNnT { #4 } = { -1 }
      { \bool_set_true:N \l_@@_final_open_bool }
    }
    {
      \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
      {
        \int_compare:nNnT { #4 } = { 1 }
        { \bool_set_true:N \l_@@_final_open_bool }
      }
    }
  }
}
\bool_if:NTF \l_@@_final_open_bool
{
  \int_sub:Nn \l_@@_final_i_int { #3 }
  \int_sub:Nn \l_@@_final_j_int { #4 }
  \bool_set_true:N \l_@@_stop_loop_bool
}
{
  \cs_if_exist:cTF
  {
    @@ _ dotted _
    \int_use:N \l_@@_final_i_int -
```

```

\int_use:N \l_@@_final_j_int
}
{
\int_sub:Nn \l_@@_final_i_int { #3 }
\int_sub:Nn \l_@@_final_j_int { #4 }
\bool_set_true:N \l_@@_final_open_bool
\bool_set_true:N \l_@@_stop_loop_bool
}
{
\cs_if_exist:cTF
{
pgf @ sh @ ns @ \@@_env:
- \int_use:N \l_@@_final_i_int
- \int_use:N \l_@@_final_j_int
}
{ \bool_set_true:N \l_@@_stop_loop_bool }
{
\cs_set_nopar:cpn
{
@@ _ dotted _
\int_use:N \l_@@_final_i_int -
\int_use:N \l_@@_final_j_int
}
{ }
}
}
}
\bool_set_false:N \l_@@_stop_loop_bool
\int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
\bool_do_until:Nn \l_@@_stop_loop_bool
{
\int_sub:Nn \l_@@_initial_i_int { #3 }
\int_sub:Nn \l_@@_initial_j_int { #4 }
\bool_set_false:N \l_@@_initial_open_bool
\int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
{
\int_compare:nNnTF { #3 } = { 1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
{
\int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
}
}
{
\int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
{
\int_compare:nNnT { #4 } = { 1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
{
\int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
{
\int_compare:nNnT { #4 } = { -1 }
{ \bool_set_true:N \l_@@_initial_open_bool }
}
}
}
}
\bool_if:NNTF \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.

```

4304 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4305 {

```

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

```

4306   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4307   \l_@@_initial_i_int = #1
4308   \l_@@_initial_j_int = #2
4309   \l_@@_final_i_int = #1
4310   \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.

```

4311   \let \l_@@_stop_loop_bool \c_false_bool
4312   \bool_do_until:Nn \l_@@_stop_loop_bool
4313   {

```

We test if we are still in the matrix.

```

4314     \advance \l_@@_final_i_int by #3
4315     \advance \l_@@_final_j_int by #4
4316     \let \l_@@_final_open_bool \c_false_bool
4317     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4318       \if_int_compare:w #3 = \c_one_int
4319         \let \l_@@_final_open_bool \c_true_bool
4320       \else:
4321         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4322           \let \l_@@_final_open_bool \c_true_bool
4323         \fi:
4324       \fi:
4325     \else:
4326       \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4327         \if_int_compare:w #4 = -1
4328           \let \l_@@_final_open_bool \c_true_bool
4329         \fi:
4330       \else:
4331         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4332           \if_int_compare:w #4 = \c_one_int
4333             \let \l_@@_final_open_bool \c_true_bool
4334           \fi:
4335         \fi:
4336       \fi:
4337     \fi:
4338     \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.

```

4339     {

```

We do a step backwards.

```

4340         \advance \l_@@_final_i_int by - #3
4341         \advance \l_@@_final_j_int by - #4
4342         \let \l_@@_stop_loop_bool \c_true_bool
4343     }

```

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`.

```

4344     {
4345       \cs_if_exist:cTF
4346       {
4347         @@ _ dotted _
4348         \int_use:N \l_@@_final_i_int -
4349         \int_use:N \l_@@_final_j_int
4350       }
4351       {
4352         \advance \l_@@_final_i_int by - #3
4353         \advance \l_@@_final_j_int by - #4
4354         \let \l_@@_final_open_bool \c_true_bool
4355         \let \l_@@_stop_loop_bool \c_true_bool
4356       }
4357       {
4358         \cs_if_exist:cTF
4359         {
4360           pgf @ sh @ ns @ \@@_env:
4361           - \int_use:N \l_@@_final_i_int
4362           - \int_use:N \l_@@_final_j_int
4363         }
4364         { \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.

```

4365         {
4366             \cs_set_nopar:cpn
4367             {
4368                 @@ _ dotted _
4369                 \int_use:N \l_@@_final_i_int -
4370                 \int_use:N \l_@@_final_j_int
4371             }
4372             { }
4373         }
4374     }
4375 }
4376 }

```

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

```

4377 \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4378 \l_tmpa_int = \l_@@_col_min_int
4379 \advance \l_tmpa_int by -1
4380 \bool_do_until:Nn \l_@@_stop_loop_bool
4381 {
4382     \advance \l_@@_initial_i_int by - #3
4383     \advance \l_@@_initial_j_int by - #4
4384     \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.

```

4385 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4386 \if_int_compare:w #3 = \c_one_int
4387 \let \l_@@_initial_open_bool \c_true_bool
4388 \else:

```

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

```

4389 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4390 \let \l_@@_initial_open_bool \c_true_bool
4391 \fi:
4392 \fi:
4393 \else:
4394 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4395 \if_int_compare:w #4 = \c_one_int
4396 \let \l_@@_initial_open_bool \c_true_bool
4397 \fi:
4398 \else:
4399 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4400 \if_int_compare:w #4 = -1
4401 \let \l_@@_initial_open_bool \c_true_bool
4402 \fi:
4403 \fi:
4404 \fi:
4405 \fi:
4406 \bool_if:NTF \l_@@_initial_open_bool
4407 {
4408     \advance \l_@@_initial_i_int by #3
4409     \advance \l_@@_initial_j_int by #4
4410     \let \l_@@_stop_loop_bool \c_true_bool
4411 }
4412 {
4413     \cs_if_exist:cTF
4414     {
4415         @@ _ dotted _
4416         \int_use:N \l_@@_initial_i_int -

```

```

4417         \int_use:N \l_@@_initial_j_int
4418     }
4419     {
4420         \advance \l_@@_initial_i_int by #3
4421         \advance \l_@@_initial_j_int by #4
4422         \let \l_@@_initial_open_bool \c_true_bool
4423         \let \l_@@_stop_loop_bool \c_true_bool
4424     }
4425     {
4426         \cs_if_exist:cTF
4427         {
4428             pgf @ sh @ ns @ \@@_env:
4429             - \int_use:N \l_@@_initial_i_int
4430             - \int_use:N \l_@@_initial_j_int
4431         }
4432         { \let \l_@@_stop_loop_bool \c_true_bool }
4433         {
4434             \cs_set_nopar:cpn
4435             {
4436                 @@ _ dotted _
4437                 \int_use:N \l_@@_initial_i_int -
4438                 \int_use:N \l_@@_initial_j_int
4439             }
4440             { }
4441         }
4442     }
4443 }
4444 }

```

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.

```

4445     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4446     {
4447         { \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`.

```

4448         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4449         { \int_use:N \l_@@_final_i_int }
4450         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4451         { }
4452     }
4453 }

```

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.

```

4454 \cs_new_protected:Npn \@@_open_shorten:
4455 {
4456     \bool_if:NT \l_@@_initial_open_bool
4457     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4458     \bool_if:NT \l_@@_final_open_bool
4459     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4460 }

```

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 sub-matrices which contains the cell of row #1 and column #2. As of now, it’s only the whole array (excepted exterior rows and columns).

```

4461 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2

```

```

4462 {
4463   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4464   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4465   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4466   \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.

```

4467   \seq_if_empty:NTF \g_@@_submatrix_seq
4468   { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4469   {

```

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`.

```

4470     \seq_map_inline:Nn \g_@@_submatrix_seq
4471     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4472   }
4473 }

```

`#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.

```

4474 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4475 {
4476   \if_int_compare:w #3 > #1
4477   \else:
4478     \if_int_compare:w #1 > #5
4479     \else:
4480       \if_int_compare:w #4 > #2
4481       \else:
4482         \if_int_compare:w #2 > #6
4483         \else:
4484           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4485           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4486           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4487           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4488         \fi:
4489       \fi:
4490     \fi:
4491   \fi:
4492 }

4493 \cs_new_protected:Npn \@@_set_initial_coords:
4494 {
4495   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

4496     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4497   }
4498   \cs_new_protected:Npn \@@_set_final_coords:
4499   {
4500     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4501     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4502   }
4503   \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4504   {
4505     \pgfpointanchor
4506     {
4507       \@@_env:
4508       - \int_use:N \l_@@_initial_i_int
4509       - \int_use:N \l_@@_initial_j_int
4510     }
4511     { #1 }
4512     \@@_set_initial_coords:
4513   }
4514   \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4515   {
4516     \pgfpointanchor
4517     {
4518       \@@_env:
4519       - \int_use:N \l_@@_final_i_int
4520       - \int_use:N \l_@@_final_j_int
4521     }
4522     { #1 }
4523     \@@_set_final_coords:
4524   }
4525   \cs_new_protected:Npn \@@_open_x_initial_dim:
4526   {
4527     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4528     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4529     {
4530       \cs_if_exist:cT
4531       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4532       {
4533         \pgfpointanchor
4534         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4535         { west }
4536         \dim_set:Nn \l_@@_x_initial_dim
4537         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4538       }
4539     }

```

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

```

4540     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4541     {
4542       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4543       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4544       \dim_add:Nn \l_@@_x_initial_dim \col@sep
4545     }
4546   }
4547   \cs_new_protected:Npn \@@_open_x_final_dim:
4548   {
4549     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4550     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4551     {
4552       \cs_if_exist:cT
4553       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4554       {
4555         \pgfpointanchor
4556         { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4557         { east }
4558         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4559         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4560     }
4561 }

```

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

```

4562     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4563     {
4564         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4565         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4566         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4567     }
4568 }

```

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.

```

4569 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4570 {
4571     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4572     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4573     {
4574         \@@_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.

```

4575     \bool_if:NT \g_@@_aux_found_bool
4576     {
4577         {
4578             \@@_open_shorten:
4579             \int_if_zero:nTF { #1 }
4580             { \color { nicematrix-first-row } }
4581         }

```

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.

```

4582         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4583         { \color { nicematrix-last-row } }
4584     }
4585     \keys_set:nn { nicematrix / xdots } { #3 }
4586     \@@_color:o \l_@@_xdots_color_tl
4587     \@@_actually_draw_Ldots:
4588 }
4589 }
4590 }
4591 }

```

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Hdotsfor`.

```

4592 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4593 {
4594   \bool_if:NTF \l_@@_initial_open_bool
4595   {
4596     \@@_open_x_initial_dim:
4597     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4598     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4599   }
4600   { \@@_set_initial_coords_from_anchor:n { base-east } }
4601   \bool_if:NTF \l_@@_final_open_bool
4602   {
4603     \@@_open_x_final_dim:
4604     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4605     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4606   }
4607   { \@@_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.

```

4608   \bool_lazy_all:nTF
4609   {
4610     \l_@@_initial_open_bool
4611     \l_@@_final_open_bool
4612     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4613   }
4614   {
4615     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4616     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4617   }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4618   {
4619     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4620     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4621   }
4622   \@@_draw_line:
4623 }

```

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.

```

4624 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4625 {
4626   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4627   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4628   {
4629     \@@_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.

```

4630   \bool_if:NT \g_@@_aux_found_bool
4631   {
4632     {
4633       \@@_open_shorten:
4634       \int_if_zero:nTF { #1 }
4635       { \color { nicematrix-first-row } }
4636     }

```

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.

```

4637   \int_compare:nNnT { #1 } = \l_@@_last_row_int

```

```

4638         { \color { nicematrix-last-row } }
4639     }
4640     \keys_set:nn { nicematrix / xdots } { #3 }
4641     \@@_color:o \l_@@_xdots_color_tl
4642     \@@_actually_draw_Cdots:
4643 }
4644 }
4645 }
4646 }

```

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.`

```

4647 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4648 {
4649     \bool_if:NTF \l_@@_initial_open_bool
4650         \@@_open_x_initial_dim:
4651         { \@@_set_initial_coords_from_anchor:n { mid-east } }
4652     \bool_if:NTF \l_@@_final_open_bool
4653         \@@_open_x_final_dim:
4654         { \@@_set_final_coords_from_anchor:n { mid-west } }
4655     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4656     {
4657         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4658         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4659         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4660         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4661         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4662     }
4663     {
4664         \bool_if:NT \l_@@_initial_open_bool
4665         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4666         \bool_if:NT \l_@@_final_open_bool
4667         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4668     }
4669     \@@_draw_line:
4670 }
4671 \cs_new_protected:Npn \@@_open_y_initial_dim:
4672 {
4673     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4674     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4675     {
4676         \cs_if_exist:cT
4677         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4678         {
4679             \pgfpointanchor
4680             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4681             { north }
4682             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4683             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4684         }
4685     }
4686     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4687     {

```

```

4688 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4689 \dim_set:Nn \l_@@_y_initial_dim
4690 {
4691   \fp_to_dim:n
4692   {
4693     \pgf@y
4694     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4695   }
4696 }
4697 }
4698 }
4699 \cs_new_protected:Npn \@@_open_y_final_dim:
4700 {
4701   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4702   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4703   {
4704     \cs_if_exist:cT
4705     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4706     {
4707       \pgfpointanchor
4708       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4709       { south }
4710       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4711       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4712     }
4713   }
4714   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4715   {
4716     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4717     \dim_set:Nn \l_@@_y_final_dim
4718     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4719   }
4720 }

```

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.

```

4721 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4722 {
4723   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4724   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4725   {
4726     \@@_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.

```

4727   \bool_if:NT \g_@@_aux_found_bool
4728   {
4729     {
4730       \@@_open_shorten:
4731       \int_if_zero:nTF { #2 }
4732       { \color { nicematrix-first-col } }
4733       {
4734         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4735         { \color { nicematrix-last-col } }
4736       }
4737       \keys_set:nn { nicematrix / xdots } { #3 }
4738       \@@_color:o \l_@@_xdots_color_tl
4739       \bool_if:NTF \l_@@_Vbrace_bool
4740       \@@_actually_draw_Vbrace:
4741       \@@_actually_draw_Vdots:
4742     }
4743   }
4744 }
4745 }

```

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`.

```

4746 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4747 {
4748   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4749   \@@_actually_draw_Vdots_i:
4750   \@@_actually_draw_Vdots_ii:
4751   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4752   \@@_draw_line:
4753 }

```

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

```

4754 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4755 {
4756   \@@_open_y_initial_dim:
4757   \@@_open_y_final_dim:
4758   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4759 {
4760   \@@_qpoint:n { col - 1 }
4761   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4762   \dim_sub:Nn \l_@@_x_initial_dim
4763   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4764 }
4765 {
4766   \bool_lazy_and:nnTF
4767   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4768   { \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”.

```

4769 {
4770   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4771   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4772   \dim_add:Nn \l_@@_x_initial_dim
4773   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4774 }

```

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

```

4775 {
4776   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4777   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4778   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4779   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4780 }
4781 }
4782 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).
The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```

4783 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4784 {
4785   \bool_set_false:N \l_tmpa_bool
4786   \bool_if:NF \l_@@_initial_open_bool
4787   {
4788     \bool_if:NF \l_@@_final_open_bool
4789     {
4790       \@@_set_initial_coords_from_anchor:n { south-west }
4791       \@@_set_final_coords_from_anchor:n { north-west }
4792       \bool_set:Nn \l_tmpa_bool
4793       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4794     }
4795   }

```

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

```

4796   \bool_if:NTF \l_@@_initial_open_bool
4797   {
4798     \@@_open_y_initial_dim:
4799     \@@_set_final_coords_from_anchor:n { north }
4800     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4801   }
4802   {
4803     \@@_set_initial_coords_from_anchor:n { south }
4804     \bool_if:NTF \l_@@_final_open_bool
4805     \@@_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.

```

4806   {
4807     \@@_set_final_coords_from_anchor:n { north }
4808     \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4809     {
4810       \dim_set:Nn \l_@@_x_initial_dim
4811       {
4812         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4813         \l_@@_x_initial_dim \l_@@_x_final_dim
4814       }
4815     }
4816   }
4817 }
4818 }

```

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`.

```

4819 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4820 {
4821   \bool_if:NTF \l_@@_initial_open_bool
4822   \@@_open_y_initial_dim:
4823   { \@@_set_initial_coords_from_anchor:n { south } }
4824   \bool_if:NTF \l_@@_final_open_bool
4825   \@@_open_y_final_dim:
4826   { \@@_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.

```

4827 \int_if_zero:nTF \l_@@_initial_j_int
4828 {
4829   \@@_qpoint:n { col - 1 }
4830   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4831   \dim_sub:Nn \l_@@_x_initial_dim
4832   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4833 }

```

Elsewhere, the brace must be drawn left flush.

```

4834 {
4835   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4836   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4837   \dim_add:Nn \l_@@_x_initial_dim
4838   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4839 }

```

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

```

4840 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4841 \@@_draw_line:
4842 }

```

```

4843 \cs_new:Npn \@@_colsep:
4844 { \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.

```

4845 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4846 {
4847   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4848   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4849   {
4850     \@@_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.

```

4851   \bool_if:NT \g_@@_aux_found_bool
4852   {
4853     {
4854       \@@_open_shorten:
4855       \keys_set:nn { nicematrix / xdots } { #3 }
4856       \@@_color:o \l_@@_xdots_color_tl
4857       \@@_actually_draw_Ddots:
4858     }
4859   }
4860 }
4861 }

```

The command `\@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`

- \l_@@_final_j_int
- \l_@@_final_open_bool.

```

4862 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4863 {
4864   \bool_if:NTF \l_@@_initial_open_bool
4865   {
4866     \@@_open_y_initial_dim:
4867     \@@_open_x_initial_dim:
4868   }
4869   { \@@_set_initial_coords_from_anchor:n { south-east } }
4870   \bool_if:NTF \l_@@_final_open_bool
4871   {
4872     \@@_open_x_final_dim:
4873     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4874   }
4875   { \@@_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.

```

4876   \bool_if:NT \l_@@_parallelize_diags_bool
4877   {
4878     \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).

```

4879     \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 Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4880     {
4881       \dim_gset:Nn \g_@@_delta_x_one_dim
4882       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4883       \dim_gset:Nn \g_@@_delta_y_one_dim
4884       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4885     }

```

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.

```

4886     {
4887       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4888       {
4889         \dim_set:Nn \l_@@_y_final_dim
4890         {
4891           \l_@@_y_initial_dim +
4892           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4893           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4894         }
4895       }
4896     }
4897   }
4898   \@@_draw_line:
4899 }

```

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.

```

4900 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4901 {
4902   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4903   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4904   {
4905     \@@_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.

```

4906     \bool_if:NT \g_@@_aux_found_bool
4907     {
4908     {
4909         \@@_open_shorten:
4910         \keys_set:nn { nicematrix / xdots } { #3 }
4911         \@@_color:o \l_@@_xdots_color_tl
4912         \@@_actually_draw_Iddots:
4913     }
4914     }
4915 }
4916 }

```

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`.

```

4917 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4918 {
4919     \bool_if:NTF \l_@@_initial_open_bool
4920     {
4921         \@@_open_y_initial_dim:
4922         \@@_open_x_initial_dim:
4923     }
4924     { \@@_set_initial_coords_from_anchor:n { south-west } }
4925     \bool_if:NTF \l_@@_final_open_bool
4926     {
4927         \@@_open_y_final_dim:
4928         \@@_open_x_final_dim:
4929     }
4930     { \@@_set_final_coords_from_anchor:n { north-east } }
4931     \bool_if:NT \l_@@_parallelize_diags_bool
4932     {
4933         \int_gincr:N \g_@@_iddots_int
4934         \int_compare:nNnTF \g_@@_iddots_int = 1
4935         {
4936             \dim_gset:Nn \g_@@_delta_x_two_dim
4937             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4938             \dim_gset:Nn \g_@@_delta_y_two_dim
4939             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4940         }
4941         {
4942             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4943             {
4944                 \dim_set:Nn \l_@@_y_final_dim
4945                 {
4946                     \l_@@_y_initial_dim +
4947                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4948                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4949                 }
4950             }
4951         }
4952     }
4953     \@@_draw_line:
4954 }

```

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`

```

4955 \cs_new_protected:Npn \@@_draw_line:
4956 {
4957   \pgfrememberpicturepositiononpagetrue
4958   \pgf@relevantforpicturesizefalse
4959   \bool_lazy_or:nnTF
4960     \l_@@_dotted_bool
4961     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4962     \@@_draw_standard_dotted_line:
4963     \@@_draw_unstandard_dotted_line:
4964 }

```

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.

```

4965 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4966 {
4967   \begin { scope }
4968     \@@_draw_unstandard_dotted_line:o
4969     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4970 }

```

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.

```

4971 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4972 {
4973   \@@_draw_unstandard_dotted_line:nooo
4974   { #1 }
4975   \l_@@_xdots_up_tl
4976   \l_@@_xdots_down_tl
4977   \l_@@_xdots_middle_tl
4978 }
4979 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following TikZ styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4980 \AtBeginDocument
4981 {
4982   \IfPackageLoadedT { tikz }
4983   {
4984     \tikzset
4985     {
4986       @@_node_above / .style = { sloped , above } ,
4987       @@_node_below / .style = { sloped , below } ,
4988       @@_node_middle / .style =
4989       {

```

```

4990         sloped ,
4991         inner~sep = \c_@@_innersep_middle_dim
4992     }
4993 }
4994 }
4995 }

4996 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4997 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4998     \dim_zero_new:N \l_@@_l_dim
4999     \dim_set:Nn \l_@@_l_dim
5000     {
5001         \fp_to_dim:n
5002         {
5003             sqrt
5004             (
5005                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5006                 +
5007                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5008             )
5009         }
5010     }

```

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.

```

5011     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5012     {
5013         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5014         \@@_draw_unstandard_dotted_line_i:
5015     }

```

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

```

5016     \bool_if:NT \l_@@_xdots_h_labels_bool
5017     {
5018         \tikzset
5019         {
5020             @@_node_above / .style = { auto = left } ,
5021             @@_node_below / .style = { auto = right } ,
5022             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
5023         }
5024     }
5025     \tl_if_empty:nF { #4 }
5026     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
5027     \dim_zero:N \l_tmpa_dim
5028     \dim_zero:N \l_tmpb_dim
5029     \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
5030     {

```

We test whether the brace is vertical or horizontal.

```

5031         \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5032         { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
5033         { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5034     }
5035     {

```

```

5036 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5037 {
5038 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5039 { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5040 { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5041 }
5042 }
5043 \use:e
5044 {
5045 \exp_not:N \begin { scope }
5046 [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5047 }
5048 \draw
5049 [ #1 ]
5050 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5051 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
5052 node [ @@_node_below ] { $ \scriptstyle #3 $ }
5053 node [ @@_node_above ] { $ \scriptstyle #2 $ }
5054 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5055 \end { scope }
5056 \end { scope }
5057 }
5058 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5059 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5060 {
5061 \dim_set:Nn \l_tmpa_dim
5062 {
5063 \l_@@_x_initial_dim
5064 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5065 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5066 }
5067 \dim_set:Nn \l_tmpb_dim
5068 {
5069 \l_@@_y_initial_dim
5070 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5071 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5072 }
5073 \dim_set:Nn \l_@@_tmpc_dim
5074 {
5075 \l_@@_x_final_dim
5076 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5077 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5078 }
5079 \dim_set:Nn \l_@@_tmpd_dim
5080 {
5081 \l_@@_y_final_dim
5082 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5083 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5084 }
5085 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5086 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5087 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5088 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5089 }

```

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

```

5090 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5091 {
5092 {

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

5093 \dim_zero_new:N \l_@@_l_dim
5094 \dim_set:Nn \l_@@_l_dim
5095 {
5096   \fp_to_dim:n
5097   {
5098     sqrt
5099     (
5100       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5101       +
5102       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5103     )
5104   }
5105 }

```

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.

```

5106 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5107 {
5108   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5109   \@@_draw_standard_dotted_line_i:
5110 }
5111 }
5112 \bool_lazy_all:nF
5113 {
5114   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5115   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5116   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5117 }
5118 \@@_labels_standard_dotted_line:
5119 }
5120 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5121 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5122 {

```

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

```

5123 \int_set:Nn \l_tmpa_int
5124 {
5125   \dim_ratio:nn
5126   {
5127     \l_@@_l_dim
5128     - \l_@@_xdots_shorten_start_dim
5129     - \l_@@_xdots_shorten_end_dim
5130   }
5131   \l_@@_xdots_inter_dim
5132 }

```

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

```

5133 \dim_set:Nn \l_tmpa_dim
5134 {
5135   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5136   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5137 }
5138 \dim_set:Nn \l_tmpb_dim
5139 {
5140   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5141   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5142 }

```

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.

```

5143   \dim_gadd:Nn \l_@@_x_initial_dim
5144   {
5145     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5146     \dim_ratio:nn
5147     {
5148       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5149       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5150     }
5151     { 2 \l_@@_l_dim }
5152   }
5153   \dim_gadd:Nn \l_@@_y_initial_dim
5154   {
5155     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5156     \dim_ratio:nn
5157     {
5158       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5159       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5160     }
5161     { 2 \l_@@_l_dim }
5162   }
5163   \pgf@relevantforpicturesizefalse
5164   \int_step_inline:nnn \c_zero_int \l_tmpa_int
5165   {
5166     \pgfpathcircle
5167     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5168     \l_@@_xdots_radius_dim
5169     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5170     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5171   }
5172   \pgfusepathqfill
5173 }

5174 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5175 {
5176   \pgfscope
5177   \pgftransformshift
5178   {
5179     \pgfpointlineattime { 0.5 }
5180     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5181     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5182   }
5183   \fp_set:Nn \l_tmpa_fp
5184   {
5185     atand
5186     (
5187       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5188       \l_@@_x_final_dim - \l_@@_x_initial_dim
5189     )
5190   }
5191   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5192   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5193   \tl_if_empty:NF \l_@@_xdots_middle_tl
5194   {
5195     \begin { pgfscope }
5196     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5197     \pgfnode
5198     { rectangle }
5199     { center }
5200     {
5201       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }

```

```

5202         {
5203             $ % $
5204             \scriptstyle \l_@@_xdots_middle_tl
5205             $ % $
5206         }
5207     }
5208     { }
5209     {
5210         \pgfsetfillcolor { white }
5211         \pgfusepath { fill }
5212     }
5213     \end { pgfscope }
5214 }
5215 \tl_if_empty:NF \l_@@_xdots_up_tl
5216 {
5217     \pgfnode
5218     { rectangle }
5219     { south }
5220     {
5221         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5222         {
5223             $ % $
5224             \scriptstyle \l_@@_xdots_up_tl
5225             $ % $
5226         }
5227     }
5228     { }
5229     { \pgfusepath { } }
5230 }
5231 \tl_if_empty:NF \l_@@_xdots_down_tl
5232 {
5233     \pgfnode
5234     { rectangle }
5235     { north }
5236     {
5237         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5238         {
5239             $ % $
5240             \scriptstyle \l_@@_xdots_down_tl
5241             $ % $
5242         }
5243     }
5244     { }
5245     { \pgfusepath { } }
5246 }
5247 \endpgfscope
5248 }

```

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.

```

5249 \AtBeginDocument
5250 {

```

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

```

5251 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5252 \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5253 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5254 {
5255   \int_if_zero:nTF \c@jCol
5256   { \@@_error:nn { in~first~col } { \Ldots } }
5257   {
5258     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5259     { \@@_error:nn { in~last~col } { \Ldots } }
5260     {
5261       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5262       { #1 , down = #2 , up = #3 , middle = #4 }
5263     }
5264   }
5265   \bool_if:NF \l_@@_nullify_dots_bool
5266   { \phantom { \ensuremath { \@@_old_ldots: } } }
5267   \bool_gset_true:N \g_@@_empty_cell_bool
5268 }

```

```

5269 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5270 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5271 {
5272   \int_if_zero:nTF \c@jCol
5273   { \@@_error:nn { in~first~col } { \Cdots } }
5274   {
5275     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5276     { \@@_error:nn { in~last~col } { \Cdots } }
5277     {
5278       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5279       { #1 , down = #2 , up = #3 , middle = #4 }
5280     }
5281   }
5282   \bool_if:NF \l_@@_nullify_dots_bool
5283   { \phantom { \ensuremath { \@@_old_cdots: } } }
5284   \bool_gset_true:N \g_@@_empty_cell_bool
5285 }

```

```

5286 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5287 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5288 {
5289   \int_if_zero:nTF \c@iRow
5290   { \@@_error:nn { in~first~row } { \Vdots } }
5291   {
5292     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5293     { \@@_error:nn { in~last~row } { \Vdots } }
5294     {
5295       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5296       { #1 , down = #2 , up = #3 , middle = #4 }
5297     }
5298   }
5299   \bool_if:NF \l_@@_nullify_dots_bool
5300   { \phantom { \ensuremath { \@@_old_vdots: } } }
5301   \bool_gset_true:N \g_@@_empty_cell_bool
5302 }

```

```

5303 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }

```

```

5304 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5305 {
5306   \int_case:nnF \c@iRow
5307   {
5308     0 { \@@_error:nn { in-first~row } { \Ddots } }
5309     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Ddots } }
5310   }
5311   {
5312     \int_case:nnF \c@jCol
5313     {
5314       0 { \@@_error:nn { in-first~col } { \Ddots } }
5315       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Ddots } }
5316     }
5317     {
5318       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5319       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5320       { #1 , down = #2 , up = #3 , middle = #4 }
5321     }
5322   }
5323 }
5324 \bool_if:NF \l_@@_nullify_dots_bool
5325 { \phantom { \ensuremath { \@@_old_ddots: } } }
5326 \bool_gset_true:N \g_@@_empty_cell_bool
5327 }

5328 \cs_new_protected:Npn \@@_Iddots:
5329 { \@@_collect_options:n { \@@_Iddots_i } }
5330 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5331 {
5332   \int_case:nnF \c@iRow
5333   {
5334     0 { \@@_error:nn { in-first~row } { \Iddots } }
5335     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Iddots } }
5336   }
5337   {
5338     \int_case:nnF \c@jCol
5339     {
5340       0 { \@@_error:nn { in-first~col } { \Iddots } }
5341       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Iddots } }
5342     }
5343     {
5344       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5345       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5346       { #1 , down = #2 , up = #3 , middle = #4 }
5347     }
5348   }
5349   \bool_if:NF \l_@@_nullify_dots_bool
5350   { \phantom { \ensuremath { \@@_old_iddots: } } }
5351   \bool_gset_true:N \g_@@_empty_cell_bool
5352 }
5353 }

```

End of the \AddToHook.

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

```

5354 \keys_define:nn { nicematrix / Ddots }
5355 {
5356   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5357   draw-first .value_forbidden:n = true ,
5358 }

```

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

```

5359 \cs_new_protected:Npn \@@_Hspace:
5360 {
5361   \bool_gset_true:N \g_@@_empty_cell_bool
5362   \hspace
5363 }

```

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`.

```

5364 \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`.

```

5365 \cs_new:Npn \@@_Hdotsfor:
5366 {
5367   \bool_lazy_and:nnTF
5368   { \int_if_zero_p:n \c@jCol }
5369   { \int_if_zero_p:n \l_@@_first_col_int }
5370   {
5371     \bool_if:NTF \g_@@_after_col_zero_bool
5372     {
5373       \multicolumn { 1 } { c } { }
5374       \@@_Hdotsfor_i:
5375     }
5376     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5377   }
5378   {
5379     \multicolumn { 1 } { c } { }
5380     \@@_Hdotsfor_i:
5381   }
5382 }

```

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:`).

```

5383 \AtBeginDocument
5384 {

```

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

```

5385   \cs_new_protected:Npn \@@_Hdotsfor_i:
5386   { \@@_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`).

```

5387   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5388   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5389   {
5390     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5391     {
5392       \@@_Hdotsfor:nnnn
5393       { \int_use:N \c@iRow }
5394       { \int_use:N \c@jCol }
5395       { #2 }
5396       {
5397         #1 , #3 ,
5398         down = \exp_not:n { #4 } ,
5399         up = \exp_not:n { #5 } ,
5400         middle = \exp_not:n { #6 }
5401       }
5402     }
5403     \prg_replicate:nn { #2 - 1 }

```

```

5404     {
5405         &
5406         \multicolumn { 1 } { c } { }
5407         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5408     }
5409 }
5410 }

```

```

5411 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5412 {
5413     \bool_set_false:N \l_@@_initial_open_bool
5414     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5415     \int_set:Nn \l_@@_initial_i_int { #1 }
5416     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5417     \int_compare:nNnTF { #2 } = 1
5418     {
5419         \int_set:Nn \l_@@_initial_j_int 1
5420         \bool_set_true:N \l_@@_initial_open_bool
5421     }
5422     {
5423         \cs_if_exist:cTF
5424         {
5425             pgf @ sh @ ns @ \@@_env:
5426             - \int_use:N \l_@@_initial_i_int
5427             - \int_eval:n { #2 - 1 }
5428         }
5429         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5430         {
5431             \int_set:Nn \l_@@_initial_j_int { #2 }
5432             \bool_set_true:N \l_@@_initial_open_bool
5433         }
5434     }
5435     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5436     {
5437         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5438         \bool_set_true:N \l_@@_final_open_bool
5439     }
5440     {
5441         \cs_if_exist:cTF
5442         {
5443             pgf @ sh @ ns @ \@@_env:
5444             - \int_use:N \l_@@_final_i_int
5445             - \int_eval:n { #2 + #3 }
5446         }
5447         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5448         {
5449             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5450             \bool_set_true:N \l_@@_final_open_bool
5451         }
5452     }
5453     \bool_if:NT \g_@@_aux_found_bool
5454     {
5455         {
5456             \@@_open_shorten:
5457             \int_if_zero:nTF { #1 }
5458             { \color { nicematrix-first-row } }
5459             {
5460                 \int_compare:nNnT { #1 } = \g_@@_row_total_int
5461                 { \color { nicematrix-last-row } }
5462             }

```

```

5463         \keys_set:nn { nicematrix / xdots } { #4 }
5464         \@@_color:o \l_@@_xdots_color_tl
5465         \@@_actually_draw_Ldots:
5466     }
5467 }

```

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).

```

5468     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5469     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - #1 } { } }
5470 }

```

```

5471 \AtBeginDocument
5472 {
5473     \cs_new_protected:Npn \@@_Vdotsfor:
5474     { \@@_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`).

```

5475     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5476     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5477     {
5478         \bool_gset_true:N \g_@@_empty_cell_bool
5479         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5480         {
5481             \@@_Vdotsfor:nnnn
5482             { \int_use:N \c@iRow }
5483             { \int_use:N \c@jCol }
5484             { #2 }
5485             {
5486                 #1 , #3 ,
5487                 down = \exp_not:n { #4 } ,
5488                 up = \exp_not:n { #5 } ,
5489                 middle = \exp_not:n { #6 }
5490             }
5491         }
5492     }
5493 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5494 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5495 {
5496     \bool_set_false:N \l_@@_initial_open_bool
5497     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5498     \int_set:Nn \l_@@_initial_j_int { #2 }
5499     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5500     \int_compare:nNnTF { #1 } = 1
5501     {
5502         \int_set:Nn \l_@@_initial_i_int 1
5503         \bool_set_true:N \l_@@_initial_open_bool
5504     }
5505     {
5506         \cs_if_exist:cTF
5507         {
5508             pgf @ sh @ ns @ \@@_env:

```

```

5509         - \int_eval:n { #1 - 1 }
5510         - \int_use:N \l_@@_initial_j_int
5511     }
5512     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5513     {
5514         \int_set:Nn \l_@@_initial_i_int { #1 }
5515         \bool_set_true:N \l_@@_initial_open_bool
5516     }
5517 }
5518 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5519 {
5520     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5521     \bool_set_true:N \l_@@_final_open_bool
5522 }
5523 {
5524     \cs_if_exist:cTF
5525     {
5526         pgf @ sh @ ns @ \@@_env:
5527         - \int_eval:n { #1 + #3 }
5528         - \int_use:N \l_@@_final_j_int
5529     }
5530     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5531     {
5532         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5533         \bool_set_true:N \l_@@_final_open_bool
5534     }
5535 }
5536 \bool_if:NT \g_@@_aux_found_bool
5537 {
5538     {
5539         \@@_open_shorten:
5540         \int_if_zero:nTF { #2 }
5541         { \color { nicematrix-first-col } }
5542         {
5543             \int_compare:nNnT { #2 } = \g_@@_col_total_int
5544             { \color { nicematrix-last-col } }
5545         }
5546         \keys_set:nn { nicematrix / xdots } { #4 }
5547         \@@_color:o \l_@@_xdots_color_tl
5548         \bool_if:NTF \l_@@_Vbrace_bool
5549         \@@_actually_draw_Vbrace:
5550         \@@_actually_draw_Vdots:
5551     }
5552 }

```

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).

```

5553     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5554     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5555 }

```

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

```

5556 \NewDocumentCommand \@@_rotate: { 0 { } }
5557 {
5558     \bool_gset_true:N \g_@@_rotate_bool
5559     \keys_set:nn { nicematrix / rotate } { #1 }
5560     \ignorespaces
5561 }

```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type *p* and *al*.

```

5562 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5563 \keys_define:nn { nicematrix / rotate }
5564 {
5565   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5566   c .value_forbidden:n = true ,
5567   -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5568   -90 .value_forbidden:n = true ,
5569   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5570 }

```

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).¹⁴

```

5571 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5572 {
5573   \tl_if_empty:nTF { #2 }
5574     { #1 }
5575     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5576 }
5577 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5578 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

5579 \AtBeginDocument
5580 {

```

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

```

5581   \tl_set_rescan:Nnn \l_tmpa_tl { }
5582   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5583   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5584   {
5585     {
5586       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5587       \@@_color:o \l_@@_xdots_color_tl
5588       \use:e
5589       {
5590         \@@_line_i:nn
5591         { \@@_double_int_eval:n #2 - \q_stop }

```

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

```

5592         { \@@_double_int_eval:n #3 - \q_stop }
5593     }
5594 }
5595 }
5596 }
5597 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5598 {
5599     \bool_set_false:N \l_@@_initial_open_bool
5600     \bool_set_false:N \l_@@_final_open_bool
5601     \bool_lazy_or:nnTF
5602     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5603     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5604     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

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

```

5605     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5606 }
5607 \AtBeginDocument
5608 {
5609     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5610     {

```

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:`.

```

5611         \c_@@_pgfortikzpicture_tl
5612         \@@_draw_line_iii:nn { #1 } { #2 }
5613         \c_@@_endpgfortikzpicture_tl
5614     }
5615 }

```

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

```

5616 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5617 {
5618     \pgfrememberpicturepositiononpagetrue
5619     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5620     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5621     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5622     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5623     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5624     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5625     \@@_draw_line:
5626 }

```

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_then: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_then: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.

```
5627 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5628 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }
```

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

```
5629 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5630 {
5631   \tl_gput_right:Ne \g_@@_row_style_tl
5632   {
```

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

```
5633   \exp_not:N
5634   \@@_if_row_less_than:nn
5635   { \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).

```
5636   {
5637     \exp_not:N
5638     \@@_if_col_greater_than:nn
5639     { \int_use:N \c@jCol }
5640     { \exp_not:n { #1 } \scan_stop: }
5641   }
5642 }
5643 }
5644 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5645 \keys_define:nn { nicematrix / RowStyle }
5646 {
5647   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5648   cell-space-top-limit+ .code:n =
5649     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5650   cell-space-top-limit+ .value_required:n = true ,
5651   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5652   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5653   cell-space-bottom-limit+ .code:n =
5654     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5655   cell-space-bottom-limit+ .value_required:n = true ,
5656   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5657   cell-space-limits .meta:n =
5658     {
5659       cell-space-top-limit = #1 ,
5660       cell-space-bottom-limit = #1 ,
5661     } ,
5662   cell-space-limits+ .meta:n =
5663     {
5664       cell-space-top-limit += #1 ,
5665       cell-space-bottom-limit += #1 ,
5666     } ,
5667   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5668   color .tl_set:N = \l_@@_color_tl ,
5669   color .value_required:n = true ,
5670   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5671   nb-rows .code:n =
5672     \str_if_eq:eeTF { #1 } { * }
5673     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5674     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5675   nb-rows .value_required:n = true ,
5676   fill .tl_set:N = \l_@@_fill_tl ,
5677   fill .value_required:n = true ,
```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

5678     opacity .tl_set:N = \l_@@_opacity_tl ,
5679     opacity .value_required:n = true ,
5680     rowcolor .tl_set:N = \l_@@_fill_tl ,
5681     rowcolor .value_required:n = true ,
5682     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5683     rounded-corners .default:n = 4 pt ,
5684     unknown .code:n =
5685       \@@_unknown_key:nn
5686       { nicematrix / RowStyle }
5687       { Unknown-key-for-RowStyle }
5688   }

```

```

5689 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5690 {
5691   \group_begin:
5692   \tl_clear:N \l_@@_fill_tl
5693   \tl_clear:N \l_@@_opacity_tl
5694   \tl_clear:N \l_@@_color_tl
5695   \int_set:Nn \l_@@_key_nb_rows_int 1
5696   \dim_zero:N \l_@@_rounded_corners_dim
5697   \dim_zero:N \l_tmpa_dim
5698   \dim_zero:N \l_tmpb_dim
5699   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `fill` (or its alias `rowcolor`) has been used.

```

5700   \tl_if_empty:NF \l_@@_fill_tl
5701   {
5702     \@@_add_opacity_to_fill:
5703     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5704     {

```

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

```

5705       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5706       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5707       {
5708         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5709         - *
5710       }
5711       { \dim_use:N \l_@@_rounded_corners_dim }
5712     }
5713   }
5714   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5715   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5716   {
5717     \@@_put_in_row_style:e
5718     {
5719       \@@_put_in_cell_after_hook:n
5720       {

```

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

```

5721       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5722       { \dim_use:N \l_tmpa_dim }
5723     }
5724   }
5725 }

```

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

```

5726   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5727   {
5728     \@@_put_in_row_style:e
5729     {

```

```

5730         \l_@@_put_in_cell_after_hook:n
5731         {
5732             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5733             { \dim_use:N \l_tmpb_dim }
5734         }
5735     }
5736 }

```

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

```

5737 \tl_if_empty:NF \l_@@_color_tl
5738 {
5739     \l_@@_put_in_row_style:e
5740     {
5741         \mode_leave_vertical:
5742         \l_@@_color:n { \l_@@_color_tl }
5743     }
5744 }

```

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

```

5745 \bool_if:NT \l_@@_bold_row_style_bool
5746 {
5747     \l_@@_put_in_row_style:n
5748     {
5749         \exp_not:n
5750         {
5751             \if_mode_math:
5752             $ % $
5753             \bfseries \boldmath
5754             $ % $
5755             \else:
5756             \bfseries \boldmath
5757             \fi:
5758         }
5759     }
5760 }
5761 \group_end:
5762 \g_@@_row_style_tl
5763 \ignorespaces
5764 }

```

The following command must *not* be protected.

```

5765 \cs_new:Npn \l_@@_rounded_from_row:n #1
5766 {
5767     \l_@@_exp_color_arg:No \l_@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5768     { \int_eval:n { #1 } - 1 }
5769     {
5770         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5771         - \exp_not:n { \int_use:N \c@jCol }
5772     }
5773     { \dim_use:N \l_@@_rounded_corners_dim }
5774 }

```

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).

```
5775 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5776 {
```

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.

```
5777 \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`.

```
5778 \str_if_in:nnF { #1 } { !! }
5779 {
5780 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5781 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5782 }
5783 \int_if_zero:nTF \l_tmpa_int
```

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

```
5784 {
5785 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5786 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5787 }
```

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

```
5788 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5789 }
5790 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
5791 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5792 {
5793 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5794 {
```

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

```
5795 \group_begin:
5796 \pgfsetcornersarced
5797 { \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.

```
5798 \bool_if:NTF \l_@@_hvlines_bool
5799 {
```

```

5800     \pgfpathrectanglecorners
5801     {
5802         \pgfpointadd
5803         { \@@_qpoint:n { row-1 } }
5804         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5805     }
5806     {
5807         \pgfpointadd
5808         {
5809             \@@_qpoint:n
5810             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5811         }
5812         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5813     }
5814 }
5815 {
5816     \pgfpathrectanglecorners
5817     { \@@_qpoint:n { row-1 } }
5818     {
5819         \pgfpointadd
5820         {
5821             \@@_qpoint:n
5822             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5823         }
5824         { \pgfpoint \c_zero_dim \arrayrulewidth }
5825     }
5826 }
5827 \pgfusepath { clip }
5828 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5829     }
5830 }

```

The command \@@_actually_color: will actually fill the rectangles, color by color, as specified in the sequence \g_@@_colors_seq.

```

5831 \cs_new_protected:Npn \@@_actually_color:
5832 {
5833     \pgfpicture
5834     \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.

```

5835     \@@_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).

```

5836     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5837     {
5838         \int_compare:nNnTF { ##1 } = 1
5839         {
5840             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5841             \use:c { g_@@_color _ 1 _tl }
5842             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5843         }
5844         {
5845             \pgfscope
5846             \@@_color_opacity: ##2
5847             \use:c { g_@@_color _ ##1 _tl }
5848             \tl_gclear:c { g_@@_color _ ##1 _tl }
5849             \pgfusepath { fill }
5850             \endpgfscope
5851         }
5852     }

```

```

5853 \endpgfpicture
5854 }

```

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

```

5855 \cs_new_protected:Npn \@@_color_opacity:
5856 {
5857   \peek_meaning:NTF [
5858     \@@_color_opacity:w
5859     { \@@_color_opacity:w [ ] }
5860   }

```

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.

```

5861 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5862 {
5863   \tl_clear:N \l_tmpa_tl
5864   \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

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

```

5865   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5866   \tl_if_empty:NTF \l_tmpb_tl
5867     \@declaredcolor
5868     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5869 }

```

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

```

5870 \keys_define:nn { nicematrix / color-opacity }
5871 {
5872   opacity .tl_set:N          = \l_tmpa_tl ,
5873   opacity .value_required:n = true
5874 }

```

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

```

5875 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5876 {
5877   \def \l_@@_rows_tl { #1 }
5878   \def \l_@@_cols_tl { #2 }
5879   \@@_cartesian_path:
5880 }

```

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

```

5881 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5882 {
5883   \tl_if_blank:nF { #2 }
5884   {
5885     \@@_add_to_colors_seq:en
5886     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5887     { \@@_cartesian_color:nn { #3 } { - } }
5888   }
5889 }

```

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

```

5890 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5891 {
5892   \tl_if_blank:nF { #2 }
5893   {
5894     \@@_add_to_colors_seq:en
5895     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }

```

```

5896         { \@@_cartesian_color:nn { - } { #3 } }
5897     }
5898 }

```

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

```

5899 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5900 {
5901     \tl_if_blank:nF { #2 }
5902     {
5903         \@@_add_to_colors_seq:en
5904         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5905         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5906     }
5907 }

```

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

```

5908 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5909 {
5910     \tl_if_blank:nF { #2 }
5911     {
5912         \@@_add_to_colors_seq:en
5913         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5914         { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5915     }
5916 }

```

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

```

5917 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5918 {
5919     \@@_cut_on_hyphen:w #1 \q_stop
5920     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5921     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5922     \@@_cut_on_hyphen:w #2 \q_stop
5923     \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5924     \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`.

```

5925     \@@_cartesian_path:n { #3 }
5926 }

```

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

```

5927 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5928 {
5929     \clist_map_inline:nn { #3 }
5930     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5931 }

```

```

5932 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5933 {
5934     \int_step_inline:nn \c@iRow
5935     {
5936         \int_step_inline:nn \c@jCol
5937         {
5938             \int_if_even:nTF { #####1 + ##1 }
5939             { \@@_cellcolor [ #1 ] { #2 } }
5940             { \@@_cellcolor [ #1 ] { #3 } }
5941             { ##1 - #####1 }
5942         }
5943     }
5944 }

```

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”.

```

5945 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5946 {
5947   \@@_rectanglecolor [ #1 ] { #2 }
5948   { 1 - 1 }
5949   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5950 }

5951 \keys_define:nn { nicematrix / rowcolors }
5952 {
5953   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5954   cols .tl_set:N = \l_@@_cols_tl ,
5955   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5956   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5957 }

```

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*.

```

5958 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5959 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5960   \seq_clear_new:N \l_@@_colors_seq
5961   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5962   \tl_clear_new:N \l_@@_cols_tl
5963   \tl_set:Nn \l_@@_cols_tl { - }
5964   \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).

```

5965   \int_zero_new:N \l_@@_color_int
5966   \int_set:Nn \l_@@_color_int 1
5967   \bool_if:NT \l_@@_respect_blocks_bool
5968   {

```

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`).

```

5969     % modified 2026-02-18
5970     \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5971     { \@@_not_in_exterior_p:nnnnn ##1 }
5972   }

```

#2 is the list of intervals of rows.

```

5973   \clist_map_inline:nn { #2 }
5974   {
5975     \tl_set:Nn \l_tmpa_tl { ##1 }
5976     \tl_if_in:NnTF \l_tmpa_tl { - }
5977     { \@@_cut_on_hyphen:w ##1 \q_stop }
5978     { \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.

```

5979     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5980     \int_set:Nn \l_@@_color_int
5981     { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5982     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5983     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5984     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
5985 \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).

```
5986 \bool_if:NT \l_@@_respect_blocks_bool
5987 {
5988   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5989   { \@@_intersect_our_row_p:nnnnn ###1 }
5990   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }
```

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

```
5991 }
5992 \tl_set:Ne \l_@@_rows_tl
5993 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
5994 \tl_set:Ne \l_@@_color_tl
5995 {
5996   \@@_color_index:n
5997   {
5998     \int_mod:nn
5999     { \l_@@_color_int - 1 }
6000     { \seq_count:N \l_@@_colors_seq }
6001     + 1
6002   }
6003 }
6004 \tl_if_empty:NF \l_@@_color_tl
6005 {
6006   \use:e
6007   {
6008     \@@_add_to_colors_seq:nn
6009     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
6010     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
6011   }
6012 }
6013 \int_incr:N \l_@@_color_int
6014 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
6015 }
6016 }
6017 }
```

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.

```
6018 \cs_new:Npn \@@_color_index:n #1
6019 {
```

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

```
6020 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
6021 { \@@_color_index:n { #1 - 1 } }
6022 { \seq_item:Nn \l_@@_colors_seq { #1 } }
6023 }
```

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.

```
6024 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
6025 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

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

```
6026 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
6027 {
6028   \int_compare:nNnT { #3 } > \l_tmpb_int
6029   { \int_set:Nn \l_tmpb_int { #3 } }
6030 }
```

```

6031 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
6032 {
6033   \int_if_zero:nTF { #4 }
6034   \prg_return_false:
6035   {
6036     \int_compare:nNnTF { #2 } > \c@jCol
6037     \prg_return_false:
6038     \prg_return_true:
6039   }
6040 }

```

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

```

6041 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6042 {
6043   \int_compare:nNnTF { #1 } > \l_tmpa_int
6044   \prg_return_false:
6045   {
6046     \int_compare:nNnTF \l_tmpa_int > { #3 }
6047     \prg_return_false:
6048     \prg_return_true:
6049   }
6050 }

```

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).

```

6051 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6052 {
6053   \dim_compare:nNnTF { #1 } = \c_zero_dim
6054   {
6055     \bool_if:NTF \l_@@_nocolor_used_bool
6056     { \@@_cartesian_path_normal_ii: }
6057     {
6058       \clist_if_empty:NTF \l_@@_corners_cells_clist
6059       { \@@_cartesian_path_normal_i:n { #1 } }
6060       { \@@_cartesian_path_normal_ii: }
6061     }
6062   }
6063   { \@@_cartesian_path_normal_i:n { #1 } }
6064 }

```

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.

```

6065 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6066 {
6067   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6068   \clist_map_inline:Nn \l_@@_cols_tl
6069   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

6070     \def \l_tmpa_tl { ##1 }
6071     \tl_if_in:NnTF \l_tmpa_tl { - }
6072     { \@@_cut_on_hyphen:w ##1 \q_stop }
6073     { \def \l_tmpb_tl { ##1 } }
6074     \tl_if_empty:NTF \l_tmpa_tl
6075     { \def \l_tmpa_tl { 1 } }
6076     {

```

```

6077     \str_if_eq:eeT \l_tmpa_tl { * }
6078     { \def \l_tmpa_tl { 1 } }
6079 }
6080 \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6081 { \@@_error:n { Invalid~col~number } }
6082 \tl_if_empty:NTF \l_tmpb_tl
6083 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6084 {
6085     \str_if_eq:eeT \l_tmpb_tl { * }
6086     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6087 }
6088 \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6089 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6090 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6091 \@@_qpoint:n { col - \l_tmpa_tl }
6092 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6093 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6094 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6095 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6096 \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.

```

6097 \clist_map_inline:Nn \l_@@_rows_tl
6098 {
6099     \def \l_tmpa_tl { #####1 }
6100     \tl_if_in:NnTF \l_tmpa_tl { - }
6101     { \@@_cut_on_hyphen:w #####1 \q_stop }
6102     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6103     \tl_if_empty:NTF \l_tmpa_tl
6104     { \def \l_tmpa_tl { 1 } }
6105     {
6106         \str_if_eq:eeT \l_tmpa_tl { * }
6107         { \def \l_tmpa_tl { 1 } }
6108     }
6109     \tl_if_empty:NTF \l_tmpb_tl
6110     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6111     {
6112         \str_if_eq:eeT \l_tmpb_tl { * }
6113         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6114     }
6115     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6116     { \@@_error:n { Invalid~row~number } }
6117     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6118     { \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`.

```

6119 \cs_if_exist:cF
6120 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6121 {
6122     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6123     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6124     \@@_qpoint:n { row - \l_tmpa_tl }
6125     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6126     \pgfpathrectanglecorners
6127     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6128     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6129 }
6130 }
6131 }
6132 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6133 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6134 {
6135   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6136   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6137   \clist_map_inline:Nn \l_@@_cols_tl
6138   {
6139     \@@_qpoint:n { col - ##1 }
6140     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6141     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6142     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6143     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6144     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6145     \clist_map_inline:Nn \l_@@_rows_tl
6146     {
6147       \@@_if_in_corner:nF { #####1 - ##1 }
6148       {
6149         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6150         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6151         \@@_qpoint:n { row - #####1 }
6152         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6153         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6154         {
6155           \pgfpathrectanglecorners
6156           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6157           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6158         }
6159       }
6160     }
6161   }
6162 }

```

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`).

```

6163 \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

```

6164 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6165 {
6166   \bool_set_true:N \l_@@_nocolor_used_bool
6167   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6168   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6169   \clist_map_inline:Nn \l_@@_rows_tl
6170   {
6171     \clist_map_inline:Nn \l_@@_cols_tl
6172     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6173   }
6174 }

```

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.

```

6175 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6176 {
6177   \clist_set_eq:NN \l_tmpa_clist #1

```

```

6178 \clist_clear:N #1
6179 \clist_map_inline:Nn \l_tmpa_clist
6180 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6181 \def \l_tmpa_tl { ##1 }
6182 \tl_if_in:NnTF \l_tmpa_tl { - }
6183 { \@@_cut_on_hyphen:w ##1 \q_stop }
6184 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6185 \bool_lazy_or:nnT
6186 { \str_if_eq_p:ee \l_tmpa_tl { * } }
6187 { \tl_if_blank_p:o \l_tmpa_tl }
6188 { \def \l_tmpa_tl { 1 } }
6189 \bool_lazy_or:nnT
6190 { \str_if_eq_p:ee \l_tmpb_tl { * } }
6191 { \tl_if_blank_p:o \l_tmpb_tl }
6192 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6193 \int_compare:nNnT \l_tmpb_tl > { #2 }
6194 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6195 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6196 { \clist_put_right:Nn #1 { #####1 } }
6197 }
6198 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6199 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6200 {
6201 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6202 {

```

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`).

```

6203 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6204 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6205 }
6206 \ignorespaces
6207 }

6208 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6209 { \@@_error:n { cellcolor~in~Block } }
6210 % \end{macrocode}
6211 %
6212 % \begin{macrocode}
6213 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6214 { \@@_error:n { rowcolor~in~Block } }
6215 % \end{macrocode}
6216 %
6217 % \bigskip
6218 % The following command will be linked to |\rowcolor| in the tabular.
6219 % \begin{macrocode}
6220 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6221 {
6222 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6223 {
6224 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6225 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6226 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6227 }
6228 \ignorespaces
6229 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by currying).

```

6230 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6231 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to \rowlistcolors in the tabular.

```

6232 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6233 {

```

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.

```

6234 \seq_gclear:N \g_tmpa_seq
6235 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6236 { \@@_rowlistcolors_tabular:nnnn #1 }
6237 \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.

```

6238 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6239 {
6240 { \int_use:N \c@iRow }
6241 { \exp_not:n { #1 } }
6242 { \exp_not:n { #2 } }
6243 { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6244 }
6245 \ignorespaces
6246 }

```

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).

```

6247 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6248 {
6249 \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).

```

6250 { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6251 {
6252 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6253 {
6254 \@@_rowlistcolors
6255 [ \exp_not:n { #2 } ]
6256 { #1 - \int_eval:n { \c@iRow - 1 } }
6257 { \exp_not:n { #3 } }
6258 [ \exp_not:n { #4 } ]
6259 }
6260 }
6261 }

```

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.

```

6262 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:

```

```

6263 {
6264   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6265     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6266   \seq_gclear:N \g_@@_rowlistcolors_seq
6267 }

6268 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6269 {
6270   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6271     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6272 }

```

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.

```

6273 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6274 {

```

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").

```

6275   \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6276   {
6277     \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).

```

6278     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6279     {
6280       \exp_not:N \columncolor [ #1 ]
6281       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6282     }
6283   }
6284 }

```

```

6285 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6286 {
6287   \clist_map_inline:nn { #1 }
6288   {
6289     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6290       { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6291     \columncolor { nocolor } { ##1 }
6292   }
6293 }

6294 \cs_new_protected:Npn \@@_EmptyRow:n #1
6295 {
6296   \clist_map_inline:nn { #1 }
6297   {
6298     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6299       { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6300     \rowcolor { nocolor } { ##1 }
6301   }
6302 }

```

The following command must *not* be protected.

```

6303 \cs_new:Npn \@@_FalseRow
6304 {
6305   \noalign
6306     { \skip_vertical:n { -\normalbaselineskip + \l_@@_width_of_false_dim } }

```

```

6307 \rowcolor { nocolor }
6308 \tl_gput_right:Ne \g_nicematrix_code_before_tl
6309 { \exp_not:N \EmptyRow { \arabic{iRow} } }
6310 \\
6311 }

```

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`.

```

6312 \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`.

```

6313 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6314 {
6315   \int_if_zero:nTF \l_@@_first_col_int
6316   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6317   {
6318     \int_if_zero:nTF \c@jCol
6319     {
6320       \int_compare:nNnF \c@iRow = { -1 }
6321       {
6322         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6323         { #1 }
6324       }
6325     }
6326     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6327   }
6328 }

```

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.

```

6329 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6330 {
6331   \int_if_zero:nF \c@iRow
6332   {
6333     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6334     {
6335       \int_compare:nNnT \c@jCol > \c_zero_int
6336       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6337     }
6338   }
6339 }

```

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`.

```

6340 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6341 {
6342   \IfPackageLoadedTF { tikz }
6343   {
6344     \IfPackageLoadedTF { booktabs }
6345     { #2 }
6346     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6347   }
6348   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6349 }

6350 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6351 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6352 \cs_new:Npn \@@_TopRule_i:
6353 {
6354   \noalign \bgroup
6355   \peek_meaning:NTF [
6356   { \@@_TopRule_ii: }
6357   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6358 }

6359 \NewDocumentCommand \@@_TopRule_ii: { o }
6360 {
6361   \tl_gput_right:Ne \g_@@_rules_tl
6362   {
6363     \@@_draw_hrulerule:n
6364     {
6365       position = \int_eval:n { \c@iRow + 1 } ,
6366       tikz =
6367       {
6368         line-width = #1 ,
6369         yshift = 0.25 \arrayrulewidth ,
6370         shorten-< = - 0.5 \arrayrulewidth
6371       } ,
6372       total-width = #1
6373     }
6374   }
6375   \skip_vertical:n { \belowrulesep + #1 }
6376   \egroup
6377 }

6378 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6379 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6380 \cs_new:Npn \@@_BottomRule_i:
6381 {
6382   \noalign \bgroup
6383   \peek_meaning:NTF [
6384   { \@@_BottomRule_ii: }
6385   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6386 }

6387 \NewDocumentCommand \@@_BottomRule_ii: { o }
6388 {
6389   \tl_gput_right:Ne \g_@@_rules_tl
6390   {
6391     \@@_draw_hrulerule:n
6392     {
6393       position = \int_eval:n { \c@iRow + 1 } ,
6394       tikz =
6395       {
6396         line-width = #1 ,

```

```

6397         yshift = 0.25 \arrayrulewidth ,
6398         shorten~< = - 0.5 \arrayrulewidth
6399     } ,
6400     total-width = #1 ,
6401 }
6402 }
6403 \skip_vertical:N \aboverulesep
6404 \@@_create_row_node_i:
6405 \skip_vertical:n { #1 }
6406 \egroup
6407 }
6408 \NewExpandableDocumentCommand { \@@_MidRule } { }
6409 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6410 \cs_new:Npn \@@_MidRule_i:
6411 {
6412     \noalign \bgroup
6413     \peek_meaning:NTF [
6414         { \@@_MidRule_ii: }
6415         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6416     ]
6417 \NewDocumentCommand \@@_MidRule_ii: { o }
6418 {
6419     \skip_vertical:N \aboverulesep
6420     \@@_create_row_node_i:
6421     \tl_gput_right:Ne \g_@@_rules_tl
6422     {
6423         \@@_draw_hrule:n
6424         {
6425             position = \int_eval:n { \c@iRow + 1 } ,
6426             tikz =
6427             {
6428                 line-width = #1 ,
6429                 yshift = 0.25 \arrayrulewidth ,
6430                 shorten~< = - 0.5 \arrayrulewidth
6431             } ,
6432             total-width = #1 ,
6433         }
6434     }
6435     \skip_vertical:n { \belowrulesep + #1 }
6436     \egroup
6437 }

```

General system for drawing rules

```

6438 \AtBeginDocument
6439 {
6440     \cs_new_protected:Npe \@@_draw_rules:
6441     {
6442         \c_@@_pgfortikzpicture_tl
6443         \@@_draw_rules_i:
6444         \c_@@_endpgfortikzpicture_tl
6445     }
6446 }
6447 \cs_new_protected:Npn \@@_draw_rules_i:
6448 {
6449     \bool_lazy_all:nT
6450     {
6451         { \clist_if_empty_p:N \l_@@_corners_clist }
6452         { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6453         { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6454         { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }

```

```

6455     { ! \l_@@_fix_vertex_bool }
6456   }
6457   {
6458     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6459     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6460   }
6461   \pgfrememberpicturepositiononpagetrue
6462   \pgf@relevantforpicturesizefalse
6463   \pgfsetrectcap
6464   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6465   \CT@arc@
6466   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6467   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6468   \g_@@_rules_tl
6469 }

```

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”.

```

6470 \keys_define:nn { nicematrix / rules-after }
6471 {
6472   position .int_set:N = \l_@@_position_int ,
6473   start .int_set:N = \l_@@_start_int ,
6474   start .initial:n = 1 ,
6475   end .int_set:N = \l_@@_end_int ,
6476   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6477   dotted .code:n =
6478     \bool_set_true:N \l_@@_dotted_bool
6479     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6480     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6481   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]`.

```

6482   color .code:n =
6483     \@@_set_CTarc:n { #1 }
6484     \CT@arc@
6485     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6486   color .value_required:n = true ,
6487   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6488   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
  }
}

6489 total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6490 unknown .code:n =
6491   \@@_unknown_key:nn
6492     { nicematrix / rules-after }
6493     { Unknown-key-for-a-rule } ,
6494 tikz .value_required:n = true
6495 }

```

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.

```

6496 \AtBeginDocument
6497 {
6498   \IfPackageLoadedTF { tikz }
6499   {
6500     \keys_define:nn { nicematrix / rules-after }
6501     {
6502       tikz .code:n =
6503         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6504         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6505         \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6506       dashed .meta:n =
6507       {
6508         tikz = nicematrix / dashed ,
6509         total-width = \pgflinewidth
6510       }
6511     }
6512   }
6513   {
6514     \keys_define:nn { nicematrix / rules-after }
6515     { tikz .code:n = \@@_error:n { tikz~without~tikz } }
6516   }
6517 }

6518 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6519 {
6520   \int_set:Nn \l_@@_multiplicity_int { #1 }
6521   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6522   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6523 }

6524 \AtBeginDocument
6525 {
6526   \IfPackageLoadedT { tikz }
6527   {
6528     \tikzset
6529     {
6530       nicematrix / dashed / .style =
6531       {
6532         dash-pattern = on~4~pt~off~2pt ,
6533         line-cap = butt
6534       }
6535     }
6536     \cs_if_exist:cT { tikz@library@decorations@loaded }
6537     { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6538   }
6539 }

```

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.

```

6540 \cs_new_protected:Npn \@@_draw_vrule:n #1
6541 {

```

The group is for the options.

```

6542 {
6543   \int_set_eq:NN \l_@@_end_int \c@iRow
6544   \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).

```

6545     \int_compare:nNt \l_@@_position_int < { \c@jCol + 2 }
6546     \socket_use:n { nicematrix / draw-vrule }
6547   }
6548 }

```

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.

```

6549 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6550 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6551 {
6552   \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`.

```

6553   \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).

```

6554   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6555   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6556   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6557   \l_tmpa_tl
6558   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6559     \@@_test_v:
6560     \bool_if:NTF \g_tmpa_bool
6561     {
6562       \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.

```

6563       { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6564       { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6565     }
6566     {
6567       \int_compare:nNt \l_@@_segment_start_int > \c_zero_int
6568       {
6569         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6570       \socket_use:n { nicematrix / vsegment }
6571       \int_zero:N \l_@@_segment_start_int
6572     }
6573   }
6574 }
6575 \int_compare:nNt \l_@@_segment_start_int > \c_zero_int
6576 {
6577   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6578   \socket_use:n { nicematrix / vsegment }
6579 }
6580 \group_end:
6581 }

```

```

6582 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6583 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6584 {
6585   \group_begin:
6586   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6587   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6588   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6589   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6590   \socket_use:n { nicematrix / vsegment }
6591   \group_end:
6592 }

```

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.

```

6593 \cs_new_protected:Npn \@@_test_v:
6594 {
6595   \bool_gset_true:N \g_tmpa_bool
6596   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6597   \bool_if:NT \g_tmpa_bool
6598   {
6599     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6600     { \@@_test_vline_in_block:nnnnn ##1 }
6601     \bool_if:NT \g_tmpa_bool
6602     {
6603       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6604       { \@@_test_vline_in_block:nnnnn ##1 }
6605       \bool_if:NT \g_tmpa_bool
6606       {
6607         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6608         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6609       }
6610     }
6611   }
6612 }
6613 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6614 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6615 {
6616   {
6617     \int_compare:nNnTF \l_@@_position_int > \c@jCol
6618     { \bool_set_false:N \l_tmpa_bool }
6619     {
6620       \@@_test_h:
6621       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6622     }
6623     \int_compare:nNnTF \l_@@_position_int = 1
6624     { \bool_gset_false:N \g_tmpa_bool }
6625     {
6626       \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6627       \@@_test_h:
6628     }
6629     \bool_gset:Nn \g_tmpa_bool
6630     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6631   }
6632   \bool_if:NT \g_tmpa_bool
6633   {
6634     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6635     \socket_use:n { nicematrix / vsegment }
6636     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6637   }
6638 }

```

```

6639 \cs_new_protected:Npn \@@_test_in_corner_v:
6640 {
6641   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6642   {
6643     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6644     { \bool_set_false:N \g_tmpa_bool }
6645   }
6646   {
6647     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6648     {
6649       \int_compare:nNnTF \l_tmpb_tl = 1
6650       { \bool_set_false:N \g_tmpa_bool }
6651       {
6652         \@@_if_in_corner:nT
6653         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6654         { \bool_set_false:N \g_tmpa_bool }
6655       }
6656     }
6657   }
6658 }

```

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`.

```

6659 \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`.

```

6660 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6661 {
6662   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6663   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6664   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6665   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6666   \pgfusepathqstroke
6667 }
6668 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6669 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6670 {
6671   \dim_add:Nn \l_@@_x_initial_dim
6672   {
6673     - 0.5 \l_@@_rule_width_after_dim
6674     +
6675     ( \arrayrulewidth * \l_@@_multiplicity_int
6676       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6677   }
6678   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6679   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6680   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6681   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6682   \bool_lazy_and:nnT
6683   { \cs_if_exist_p:N \CT@drsc@ }
6684   { ! \tl_if_blank_p:o \CT@drsc@ }
6685   {
6686     {

```

```

6687 \CT@drsc@
6688 \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6689 \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6690 \dim_set:Nn \l_@@_tmpd_dim
6691 {
6692   \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6693   * ( \l_@@_multiplicity_int - 1 )
6694 }
6695 \pgfpathrectanglecorners
6696 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6697 { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6698 \pgfusepath { fill }
6699 }
6700 }
6701 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6702 \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6703 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6704 {
6705   \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6706   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6707   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6708 }
6709 \pgfusepathqstroke
6710 }

6711 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6712 {
6713   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6714   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6715   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6716   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6717   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6718   \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`

```

6719 \@@_draw_line:
6720 }

6721 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6722 {
6723   {

```

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).

```

6724 \tl_if_empty:NF \l_@@_rule_color_tl
6725 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6726 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6727 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6728 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6729 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6730     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6731     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6732     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6733     ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6734   }
6735 }

```

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).

```

6736 \cs_new_protected:Npn \@@_draw_key_vlines:
6737 {
6738 {
6739   \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6740   \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currrification in the following code.

```

6741   \str_if_eq:eeTF \l_@@_vlines_clist { all }
6742   { \@@_lines_step_inline:n \c@jCol }
6743   { \clist_map_inline:Nn \l_@@_vlines_clist }
6744   {
6745     \int_set:Nn \l_@@_position_int { ##1 }
6746     \socket_use:n { nicematrix / draw-vrule }
6747   }
6748 }
6749 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6750 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6751 {
6752   \int_step_inline:nnn
6753   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6754   {
6755     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6756     { #1 }
6757     { \int_eval:n { #1 + 1 } }
6758   }
6759 }

```

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.

```

6760 \cs_new_protected:Npn \@@_draw_hrule:n #1
6761 {
6762 {
6763   \int_set_eq:NN \l_@@_end_int \c@jCol
6764   \keys_set_known:nn { nicematrix / rules-after } { #1 }
6765   \socket_use:nn { nicematrix / draw-hrule }
6766 }
6767 }

```

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.

```

6768 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6769 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6770 {
6771   \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`.

```

6772   \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).

```

6773   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6774   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6775   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6776     \l_tmpb_tl
6777   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6778     \@@_test_h:
6779     \bool_if:NTF \g_tmpa_bool
6780     {
6781       \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.

```

6782         { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6783         { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6784     }
6785     {
6786       \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6787       {
6788         \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

6789         \socket_use:n { nicematrix / hsegment }
6790         \int_zero:N \l_@@_segment_start_int
6791     }
6792 }
6793 }
6794 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6795 {
6796   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6797   \socket_use:n { nicematrix / hsegment }
6798 }
6799 \group_end:
6800 }

6801 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6802 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6803 {
6804   \group_begin:
6805   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6806   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6807   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6808   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6809   \socket_use:n { nicematrix / hsegment }
6810   \group_end:
6811 }

```

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.

```

6812 \cs_new_protected:Npn \@@_test_h:
6813 {
6814   \bool_gset_true:N \g_tmpa_bool
6815   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6816   \bool_if:NT \g_tmpa_bool
6817   {
6818     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6819     { \@@_test_hline_in_block:nnnnn ##1 }
6820     \bool_if:NT \g_tmpa_bool
6821     {
6822       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6823       { \@@_test_hline_in_block:nnnnn ##1 }
6824       \bool_if:NT \g_tmpa_bool
6825       {
6826         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6827         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6828       }
6829     }
6830   }
6831 }

6832 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6833 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6834 {
6835   {
6836     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6837     { \bool_set_false:N \l_tmpa_bool }
6838     {
6839       \@@_test_v:
6840       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6841     }
6842     \int_compare:nNnTF \l_@@_position_int = 1
6843     { \bool_gset_false:N \g_tmpa_bool }
6844     {
6845       \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6846       \@@_test_v:
6847     }
6848     \bool_gset:Nn \g_tmpa_bool
6849     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6850   }
6851   \bool_if:NT \g_tmpa_bool
6852   {
6853     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6854     \socket_use:n { nicematrix / hsegment }
6855     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6856   }
6857 }

6858 \cs_new_protected:Npn \@@_test_in_corner_h:
6859 {
6860   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6861   {
6862     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6863     { \bool_set_false:N \g_tmpa_bool }
6864   }
6865   {
6866     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6867     {

```

```

6868         \int_compare:nNnTF \l_tmpa_tl = 1
6869         { \bool_set_false:N \g_tmpa_bool }
6870         {
6871             \@@_if_in_corner:nT
6872             { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6873             { \bool_set_false:N \g_tmpa_bool }
6874         }
6875     }
6876 }
6877 }

```

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`.

```

6878 \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`.

```

6879 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6880 {
6881     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6882     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6883     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6884     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6885     \pgfusepathqstroke
6886 }
6887 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
6888 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6889 {
6890     \dim_add:Nn \l_@@_y_initial_dim
6891     {
6892         - 0.5 \l_@@_rule_width_after_dim
6893         +
6894         ( \arrayrulewidth * \l_@@_multiplicity_int
6895           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6896     }
6897     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6898     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6899     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6900     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6901     \bool_lazy_and:nnT
6902     { \cs_if_exist_p:N \CT@drsc@ }
6903     { ! \tl_if_blank_p:o \CT@drsc@ }
6904     {
6905         {
6906             \CT@drsc@
6907             \dim_set:Nn \l_@@_tmpd_dim
6908             {
6909                 \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6910                 * ( \l_@@_multiplicity_int - 1 )
6911             }
6912             \pgfpathrectanglecorners
6913             { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6914             { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6915             \pgfusepathqfill

```

```

6916     }
6917   }
6918   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6919   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6920   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6921   {
6922     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6923     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6924     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6925   }
6926   \pgfusepathqstroke
6927 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6928 \socket_new_plugin:nnn { nicematrix / hsegment } { dotted }
6929 {
6930   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6931   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6932   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6933   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6934   \int_compare:nNnT \l_@@_segment_start_int = 1
6935   {
6936     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6937     \bool_if:NF \g_@@_delims_bool
6938     { \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.

```

6939     \tl_if_eq:NnF \g_@@_left_delim_tl (
6940       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6941     )
6942     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6943     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6944     \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6945     {
6946       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6947       \bool_if:NF \g_@@_delims_bool
6948       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6949       \tl_if_eq:NnF \g_@@_right_delim_tl )
6950       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6951     }

```

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`

```

6952   \@@_draw_line:
6953 }

```

```

6954 \socket_new_plug:nmn { nicematrix / hsegment } { tikz }
6955 {
6956   {

```

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).

```

6957   \tl_if_empty:NF \l_@@_rule_color_tl
6958   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6959   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6960   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6961   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6962   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6963   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6964   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6965   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6966   -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6967 }
6968 }

```

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).

```

6969 \cs_new_protected:Npn \@@_draw_key_hlines:
6970 {
6971   {
6972     \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6973     \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6974   \str_if_eq:eeTF \l_@@_hlines_clist { all }
6975   { \@@_lines_step_inline:n \c@iRow }
6976   { \clist_map_inline:Nn \l_@@_hlines_clist }
6977   {
6978     \int_set:Nn \l_@@_position_int { ##1 }
6979     \socket_use:n { nicematrix / draw-hrule }
6980   }
6981 }
6982 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6983 \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.

```

6984 \cs_set:Npn \@@_Hline_i:n #1
6985 {
6986   \peek_remove_spaces:n
6987   {
6988     \peek_meaning:NTF \Hline
6989     { \@@_Hline_ii:n { #1 + 1 } }
6990     { \@@_Hline_iii:n { #1 } }
6991   }
6992 }

```

```

6993 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6994 \cs_set:Npn \@@_Hline_iii:n #1
6995 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6996 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6997 {
6998   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6999   \skip_vertical:N \l_@@_rule_width_before_dim
7000   \tl_gput_right:Ne \g_@@_rules_tl
7001   {

```

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:

```

7002   {
7003     \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
7004     \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
7005     \dim_set:Nn \l_@@_rule_width_after_dim
7006     { \dim_use:N \l_@@_rule_width_before_dim }
7007     \@@_draw_hrule:n { #2 }
7008   }
7009 }
7010 \egroup
7011 }

```

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`).

```

7012 \cs_new_protected:Npn \@@_custom_line:n #1
7013 {
7014   \str_clear:N \l_@@_letter_str
7015   \str_clear:N \l_@@_command_str
7016   \str_clear:N \l_@@_ccommand_str
7017   \tl_clear_new:N \l_@@_other_keys_tl
7018   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
7019   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
7020   {
7021     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

7022     \str_set:Ne \l_@@_command_str
7023     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
7024   }
7025   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
7026   {
7027     \str_set:Ne \l_@@_ccommand_str
7028     { \str_tail:N \l_@@_ccommand_str }
7029     \str_set:Ne \l_@@_ccommand_str
7030     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
7031   }

```

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.

```

7032 \bool_lazy_all:nTF
7033 {
7034   { \str_if_empty_p:N \l_@@_letter_str }
7035   { \str_if_empty_p:N \l_@@_command_str }
7036   { \str_if_empty_p:N \l_@@_ccommand_str }
7037 }
7038 { \@@_error:n { No~letter~and~no~command } }
7039 { \@@_custom_line_i:o \l_@@_other_keys_tl }
7040 }
7041 \keys_define:nn { nicematrix / custom-line }
7042 {
7043   letter .str_set:N = \l_@@_letter_str ,
7044   letter .value_required:n = true ,
7045   command .str_set:N = \l_@@_command_str ,
7046   command .value_required:n = true ,
7047   ccommand .str_set:N = \l_@@_ccommand_str ,
7048   ccommand .value_required:n = true
7049 }

```

```

7050 \cs_new_protected:Npn \@@_custom_line_i:n #1
7051 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7052 \bool_set_false:N \l_@@_tikz_rule_bool
7053 \bool_set_false:N \l_@@_dotted_rule_bool
7054 \bool_set_false:N \l_@@_color_bool
7055 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7056 \bool_if:NT \l_@@_tikz_rule_bool
7057 {
7058   \IfPackageLoadedF { tikz }
7059   { \@@_error:n { tikz~in~custom~line~without~tikz } }
7060   \bool_if:NT \l_@@_color_bool
7061   { \@@_error:n { color~in~custom~line~with~tikz } }
7062 }
7063 \bool_if:NT \l_@@_dotted_rule_bool
7064 {
7065   \int_compare:nNnT \l_@@_multiplicity_int > 1
7066   { \@@_error:n { key~multiplicity~with~dotted } }
7067 }
7068 \str_if_empty:NF \l_@@_letter_str
7069 {
7070   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7071   { \@@_error:n { Several~letters } }
7072   {
7073     \tl_if_in:NoTF
7074     \c_@@_forbidden_letters_str
7075     \l_@@_letter_str
7076     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
7077   }

```

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.

```

7078 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7079 { \@@_v_custom_line:nn { #1 } }
7080 }

```

```

7081     }
7082   }
7083   \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7084   \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7085 }
7086 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7087 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
7088 \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.

```

7089 \keys_define:nn { nicematrix / custom-line-bis }
7090 {
7091   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7092   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7093   color .value_required:n = true ,
7094   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7095   tikz .value_required:n = true ,
7096   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7097   dotted .value_forbidden:n = true ,
7098   dashed .meta:n =
7099   {
7100     tikz = nicematrix / dashed ,
7101     total-width = \pgflinewidth
7102   } ,
7103   total-width .code:n = { } ,
7104   total-width .value_required:n = true ,
7105   sep-color .code:n = { } ,
7106   sep-color .value_required:n = true ,
7107   unknown .code:n =
7108   \@@_unknown_key:nn
7109   { nicematrix / custom-line-bis }
7110   { Unknown-key-for-custom-line }
7111 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7112 \bool_new:N \l_@@_dotted_rule_bool
7113 \bool_new:N \l_@@_tikz_rule_bool
7114 \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).

```

7115 \keys_define:nn { nicematrix / custom-line-width }
7116 {
7117   multiplicity .int_set:N = \l_@@_tmpc_int ,
7118   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7119   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7120   \bool_set_true:N \l_@@_total_width_bool ,
7121   total-width .value_required:n = true ,
7122   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7123 }

```

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_hrule:n` (which is in the internal `\CodeAfter`).

```

7124 \cs_new_protected:Npn \@@_h_custom_line:n #1
7125 {

```

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`).

```

7126 \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7127 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7128 }

```

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_hrule:n` (which is in the internal `\CodeAfter`).

```

7129 \cs_new_protected:Npn \@@_c_custom_line:n #1
7130 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7131 \exp_args:Nc \DeclareExpandableDocumentCommand
7132 { nicematrix - \l_@@_ccommand_str }
7133 { 0 { } m }
7134 {
7135 \noalign
7136 {
7137 \@@_compute_rule_width:n { #1 , ##1 }
7138 \skip_vertical:n \l_@@_rule_width_before_dim
7139 \clist_map_inline:nn
7140 { ##2 }
7141 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7142 }
7143 }
7144 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7145 }

```

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*.

```

7146 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7147 {
7148 \tl_if_in:nnTF { #2 } { - }
7149 { \@@_cut_on_hyphen:w #2 \q_stop }
7150 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7151 \tl_gput_right:Ne \g_@@_rules_tl
7152 {
7153 \@@_draw_hrule:n
7154 {
7155 #1 ,
7156 start = \l_tmpa_tl ,
7157 end = \l_tmpb_tl ,
7158 position = \int_eval:n { \c@iRow + 1 } ,
7159 total-width = \dim_use:N \l_@@_rule_width_before_dim
7160 }
7161 }
7162 }

7163 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7164 {
7165 \bool_set_false:N \l_@@_tikz_rule_bool
7166 \bool_set_false:N \l_@@_total_width_bool
7167 \bool_set_false:N \l_@@_dotted_rule_bool
7168 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7169 \bool_if:NF \l_@@_total_width_bool
7170 {
7171 \bool_if:NTF \l_@@_dotted_rule_bool
7172 { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7173 {
7174 \bool_if:NF \l_@@_tikz_rule_bool
7175 {

```

```

7176         \dim_set:Nn \l_@@_rule_width_before_dim
7177         {
7178             \arrayrulewidth * \l_@@_tmpc_int
7179             + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7180         }
7181     }
7182 }
7183 }
7184 }

```

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

```

7185 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7186 {
7187     \str_if_eq:nnTF { #2 } { [ ] }
7188     { \@@_v_custom_line_i:nw { #1 } [ ] }
7189     { \@@_v_custom_line_ii:nn { #2 } { #1 } } }
7190 }
7191 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7192 { \@@_v_custom_line:nn { #1 , #2 } }
7193 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7194 {
7195     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7196     \tl_gput_right:Ne \g_@@_array_preamble_tl
7197     {
7198         \exp_not:N !
7199         { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7200     }
7201     \tl_gput_right:Ne \g_@@_rules_tl
7202     {

```

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.

```

7203     {
7204         \dim_set:Nn \l_@@_rule_width_after_dim
7205         { \dim_use:N \l_@@_rule_width_before_dim }
7206         \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7207         \@@_draw_vrule:n { #2 }
7208     }
7209 }
7210 \@@_rec_preamble:n #1
7211 }
7212 \@@_custom_line:n
7213 { letter = : , command = \hdottedline , ccommand = \cdottedline , dotted }

```

The key default-line

```

7214 \keys_define:nn { nicematrix / default-line }
7215 {
7216     multiplicity .int_set:N = \l_@@_multiplicity_int ,
7217     color .code:n = \bool_set_true:N \l_@@_color_bool ,
7218     color .value_required:n = true ,
7219     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,

```

```

7220 dotted .value_forbidden:n = true ,
7221 total-width .code:n = { } ,
7222 total-width .value_required:n = true ,
7223 sep-color .code:n = { } ,
7224 sep-color .value_required:n = true ,
7225 unknown .code:n =
7226   \@@_unknown_key:nn
7227     { nicematrix / default-line }
7228     { Unknown-key-for~default-line }
7229 }

7230 \AddToHook{ package / tikz / after }
7231 {
7232   \keys_define:nn { nicematrix / default-line }
7233   {
7234     tikz .code:n =
7235       \bool_set_true:N \l_@@_tikz_rule_bool
7236       \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7237     tikz .value_required:n = true ,
7238     dashed .meta:n =
7239       {
7240         tikz = nicematrix / dashed ,
7241         total-width = \pgflinewidth
7242       }
7243   }
7244   \@@_custom_line:n
7245   {
7246     letter = ; ,
7247     command = \hdashedline ,
7248     ccommand = \cdashedline ,
7249     tikz = nicematrix / dashed ,
7250     total-width = \pgflinewidth
7251   }
7252 }

7253 \cs_new_protected:Npn \@@_default_line:n #1
7254 {
7255   \bool_set_false:N \l_@@_tikz_rule_bool
7256   \bool_set_false:N \l_@@_dotted_rule_bool
7257   \bool_set_false:N \l_@@_color_bool
7258   \keys_set:nn { nicematrix / default-line } { #1 }
7259   \bool_if:NT \l_@@_tikz_rule_bool
7260   {
7261     \IfPackageLoadedF { tikz }
7262       { \@@_error:n { tikz-in~custom-line-without~tikz } }
7263     \bool_if:NT \l_@@_color_bool
7264       { \@@_error:n { color-in~custom-line-with~tikz } }
7265   }
7266   \bool_if:NT \l_@@_dotted_rule_bool
7267   {
7268     \int_compare:nNnT \l_@@_multiplicity_int > 1
7269       { \@@_error:n { key-multiplicity-with-dotted } }
7270   }
7271   \bool_if:NTF \l_@@_tikz_rule_bool
7272   {
7273     \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7274     \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7275   }
7276   {
7277     \bool_if:NTF \l_@@_dotted_rule_bool
7278     {
7279       \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7280       \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7281     }
7282     {

```

```

7283         \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7284         \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7285     }
7286 }
7287 }

```

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.

```

7288 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7289 {
7290     \int_compare:nNt \l_tmpa_tl > { #1 }
7291     {
7292         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7293         {
7294             \int_compare:nNt \l_tmpb_tl > { #2 - 1 }
7295             {
7296                 \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7297                 { \bool_gset_false:N \g_tmpa_bool }
7298             }
7299         }
7300     }
7301 }

```

The same for vertical rules.

```

7302 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7303 {
7304     \int_compare:nNt \l_tmpa_tl > { #1 - 1 }
7305     {
7306         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7307         {
7308             \int_compare:nNt \l_tmpb_tl > { #2 }
7309             {
7310                 \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7311                 { \bool_gset_false:N \g_tmpa_bool }
7312             }
7313         }
7314     }
7315 }

7316 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7317 {
7318     \int_compare:nNt \l_tmpb_tl > { #2 - 1 }
7319     {
7320         \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7321         {
7322             \int_compare:nNtTF \l_tmpa_tl = { #1 }
7323             { \bool_gset_false:N \g_tmpa_bool }
7324             {
7325                 \int_compare:nNt \l_tmpa_tl = { #3 + 1 }
7326                 { \bool_gset_false:N \g_tmpa_bool }
7327             }
7328         }
7329     }
7330 }

7331 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7332 {
7333     \int_compare:nNt \l_tmpa_tl > { #1 - 1 }
7334     {
7335         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7336         {

```

```

7337         \int_compare:nNnTF \l_tmpb_tl = { #2 }
7338         { \bool_gset_false:N \g_tmpa_bool }
7339         {
7340             \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7341             { \bool_gset_false:N \g_tmpa_bool }
7342         }
7343     }
7344 }
7345 }

```

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.

```

7346 \cs_new_protected:Npn \@@_compute_corners:
7347 {
7348     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7349     { \@@_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`).

```

7350     \clist_clear:N \l_@@_corners_cells_clist
7351     \clist_map_inline:Nn \l_@@_corners_cells_clist
7352     {
7353         \str_case:nnF { ##1 }
7354         {
7355             { NW }
7356             { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7357             { NE }
7358             { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7359             { SW }
7360             { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7361             { SE }
7362             { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7363             { N }
7364             { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7365             { S }
7366             { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7367             { E }
7368             { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7369             { W }
7370             { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7371         }
7372         { \@@_error:nn { bad~corner } { ##1 } }
7373     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7374     \clist_if_empty:NF \l_@@_corners_cells_clist
7375     {

```

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.

```

7376     \tl_gput_right:Ne \g_@@_aux_tl
7377     {
7378         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7379         { \l_@@_corners_cells_clist }

```

```

7380     }
7381   }
7382 }

7383 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7384 {
7385   \int_step_inline:nnn { #1 } { #3 }
7386   {
7387     \int_step_inline:nnn { #2 } { #4 }
7388     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7389   }
7390 }

7391 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7392 {
7393   \cs_if_exist:cTF
7394   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7395   \prg_return_true:
7396   \prg_return_false:
7397 }

```

“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.

```

7398 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7399 {

```

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.

```

7400   \bool_set_false:N \l_tmpa_bool
7401   \int_zero_new:N \l_@@_last_empty_row_int
7402   \int_set:Nn \l_@@_last_empty_row_int { #1 }
7403   \int_step_inline:nnnn { #1 } { #3 } { #5 }
7404   {
7405     \bool_lazy_or:nnTF
7406     {
7407       \cs_if_exist_p:c
7408       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7409     }
7410     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7411     { \bool_set_true:N \l_tmpa_bool }
7412     {
7413       \bool_if:NF \l_tmpa_bool
7414       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7415     }
7416   }

```

Now, you determine the last empty cell in the row of number 1.

```

7417 \bool_set_false:N \l_tmpa_bool
7418 \int_zero_new:N \l_@@_last_empty_column_int
7419 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7420 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7421 {
7422   \bool_lazy_or:nnTF
7423   {
7424     \cs_if_exist_p:c
7425     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7426   }
7427   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7428   { \bool_set_true:N \l_tmpa_bool }
7429   {
7430     \bool_if:NF \l_tmpa_bool
7431     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7432   }
7433 }

```

Now, we loop over the rows.

```

7434 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7435 {

```

We treat the row number ##1 with another loop.

```

7436 \bool_set_false:N \l_tmpa_bool
7437 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7438 {
7439   \bool_lazy_or:nnTF
7440   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7441   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7442   { \bool_set_true:N \l_tmpa_bool }
7443   {
7444     \bool_if:NF \l_tmpa_bool
7445     {
7446       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7447       \clist_put_right:Nn
7448       \l_@@_corners_cells_clist
7449       { ##1 - #####1 }
7450       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7451     }
7452   }
7453 }
7454 }
7455 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7456 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7457 \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 } ...`

```

7458 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7459 {
7460   \int_step_inline:nn { #1 }
7461   {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7462 \bool_set_false:N \l_tmpa_bool
7463 \int_step_inline:nnnn { #2 } { #3 } { #4 }
7464 {
7465   \bool_lazy_or:nnTF
7466   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 } }
7467   { \@@_if_in_block_p:nn { #####1 } { ##1 } }

```

```

7468         { \bool_set_true:N \l_tmpa_bool }
7469         {
7470             \bool_if:NF \l_tmpa_bool
7471             {
7472                 \clist_put_right:Nn
7473                 \l_@@_corners_cells_clist
7474                 { #####1 - ##1 }
7475                 \cs_set_nopar:cpn { @@ _ corner _ #####1 - ##1 } { }
7476             }
7477         }
7478     }
7479 }
7480 }

```

```

7481 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7482 {
7483     \int_step_inline:nn { #1 }
7484     {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7485         \bool_set_false:N \l_tmpa_bool
7486         \int_step_inline:nnnn { #2 } { #3 } { #4 }
7487         {
7488             \bool_lazy_or:nnTF
7489             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7490             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7491             { \bool_set_true:N \l_tmpa_bool }
7492         {
7493             \bool_if:NF \l_tmpa_bool
7494             {
7495                 \clist_put_right:Nn
7496                 \l_@@_corners_cells_clist
7497                 { ##1 - #####1 }
7498                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7499             }
7500         }
7501     }
7502 }
7503 }

```

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.

```

7504 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7505 \keys_define:nn { nicematrix / NiceMatrixBlock }
7506 {
7507     auto-columns-width .code:n =
7508     {
7509         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7510         \dim_gzero_new:N \g_@@_max_cell_width_dim
7511         \bool_set_true:N \l_@@_auto_columns_width_bool
7512     }
7513 }

```

```

7514 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7515 {
7516   \int_gincr:N \g_@@_NiceMatrixBlock_int
7517   \dim_zero:N \l_@@_columns_width_dim
7518   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7519   \bool_if:NT \l_@@_block_auto_columns_width_bool
7520   {
7521     \cs_if_exist:cT
7522     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7523     {
7524       \dim_set:Nn \l_@@_columns_width_dim
7525       {
7526         \use:c
7527         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7528       }
7529     }
7530   }
7531 }

```

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`).

```

7532 {
7533   \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.

```

7534   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7535   {
7536     \bool_if:NT \l_@@_block_auto_columns_width_bool
7537     {
7538       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7539       \iow_shipout:Ne \@mainaux
7540       {
7541         \cs_gset:cpn
7542         { @@ _ 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.

```

7543       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7544     }
7545     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7546   }
7547 }
7548 \ignorespacesafterend
7549 }

```

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).

```

7550 \cs_new_protected:Npn \@@_create_extra_nodes:
7551 {
7552   \bool_if:nTF \l_@@_medium_nodes_bool
7553   {
7554     \bool_if:NTF \l_@@_no_cell_nodes_bool
7555     { \@@_error:n { extra-nodes~with~no~cell~nodes } }
7556     {
7557       \bool_if:NTF \l_@@_large_nodes_bool

```

```

7558         \@@_create_medium_and_large_nodes:
7559         \@@_create_medium_nodes:
7560     }
7561 }
7562 {
7563     \bool_if:NT \l_@@_large_nodes_bool
7564     {
7565         \bool_if:NTF \l_@@_no_cell_nodes_bool
7566         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7567         \@@_create_large_nodes:
7568     }
7569 }
7570 }

```

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`.

```

7571 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7572 {
7573     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7574     {
7575         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7576         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7577         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7578         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7579     }
7580     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7581     {
7582         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7583         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7584         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7585         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7586     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7587     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7588     {
7589         \int_step_variable:nnNn
7590         \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.

```

7591     {
7592         \cs_if_exist:cT
7593         { 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`.

```

7594     {
7595         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7596         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7597         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7598         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7599         {
7600             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7601             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7602         }

```

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`.

```

7603         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7604         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7605         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7606         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7607         {
7608             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7609             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7610         }
7611     }
7612 }
7613 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7614 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7615 {
7616     \dim_compare:nNnT
7617     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7618     {
7619         \@@_qpoint:n { row - \@@_i: - base }
7620         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7621         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7622     }
7623 }
7624 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7625 {
7626     \dim_compare:nNnT
7627     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7628     {
7629         \@@_qpoint:n { col - \@@_j: }
7630         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7631         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7632     }
7633 }
7634 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7635 \cs_new_protected:Npn \@@_create_medium_nodes:
7636 {
7637     \pgfpicture
7638     \pgfrememberpicturepositiononpagetrue
7639     \pgf@relevantforpicturesizefalse
7640     \@@_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”.

```

7641     \tl_set:Nn \l_@@_suffix_tl { -medium }
7642     \@@_create_nodes:

```

```

7643 \endpgfpicture
7644 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁵. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7645 \cs_new_protected:Npn \@@_create_large_nodes:
7646 {
7647   \pgfpicture
7648   \pgfrememberpicturepositiononpagetrue
7649   \pgf@relevantforpicturesizefalse
7650   \@@_computations_for_medium_nodes:
7651   \@@_computations_for_large_nodes:
7652   \tl_set:Nn \l_@@_suffix_tl { - large }
7653   \@@_create_nodes:
7654   \endpgfpicture
7655 }
7656 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7657 {
7658   \pgfpicture
7659   \pgfrememberpicturepositiononpagetrue
7660   \pgf@relevantforpicturesizefalse
7661   \@@_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”.

```

7662   \tl_set:Nn \l_@@_suffix_tl { - medium }
7663   \@@_create_nodes:
7664   \@@_computations_for_large_nodes:
7665   \tl_set:Nn \l_@@_suffix_tl { - large }
7666   \@@_create_nodes:
7667   \endpgfpicture
7668 }

```

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.

```

7669 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7670 {
7671   \int_set:Nn \l_@@_first_row_int 1
7672   \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`.

```

7673   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7674   {
7675     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7676     {
7677       (
7678         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7679         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7680       )
7681       / 2
7682     }
7683     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7684     { l_@@_row _ \@@_i: _ min _ dim }
7685   }
7686   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7687 {
7688   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7689   {
7690     (
7691       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7692       \dim_use:c
7693         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7694     )
7695     / 2
7696   }
7697   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7698   { l_@@_column _ \@@_j: _ max _ dim }
7699 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7700 \dim_sub:cn
7701   { l_@@_column _ 1 _ min _ dim }
7702   \l_@@_left_margin_dim
7703 \dim_add:cn
7704   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7705   \l_@@_right_margin_dim
7706 }

```

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).

```

7707 \cs_new_protected:Npn \@@_create_nodes:
7708 {
7709   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7710   {
7711     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7712     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7713       \@@_pgf_rect_node:nnnnn
7714       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7715       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7716       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7717       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7718       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7719       \str_if_empty:NF \l_@@_name_str
7720       {
7721         \pgfnodealias
7722         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7723         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7724       }
7725     }
7726   }
7727 \int_step_inline:nn \c@iRow
7728 {
7729   \pgfnodealias
7730   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7731   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7732 }
7733 \int_step_inline:nn \c@jCol
7734 {
7735   \pgfnodealias
7736   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7737   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7738 }

```

```

7739 \pgfnodealias
7740 { \@@_env: - last - last \l_@@_suffix_tl }
7741 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol \l_@@_suffix_tl }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

7742 \seq_map_pairwise_function:NNN
7743 \g_@@_multicolumn_cells_seq
7744 \g_@@_multicolumn_sizes_seq
7745 \@@_node_for_multicolumn:nn
7746 }

7747 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7748 {
7749   \cs_set_nopar:Npn \@@_i: { #1 }
7750   \cs_set_nopar:Npn \@@_j: { #2 }
7751 }

```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

7752 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7753 {
7754   \@@_extract_coords_values: #1 \q_stop
7755   \@@_pgf_rect_node:nnnnn
7756   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7757   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7758   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7759   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7760   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7761   \str_if_empty:NF \l_@@_name_str
7762   {
7763     \pgfnodealias
7764     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7765     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7766   }
7767 }

```

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).

```

7768 \keys_define:nn { nicematrix / BlockFirstPass }
7769 {
7770   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7771           \bool_set_true:N \l_@@_p_block_bool ,
7772   j .value_forbidden:n = true ,
7773   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7774   l .value_forbidden:n = true ,
7775   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7776   r .value_forbidden:n = true ,
7777   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7778   c .value_forbidden:n = true ,

```

```

7779 L.code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7780 L.value_forbidden:n = true ,
7781 R.code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7782 R.value_forbidden:n = true ,
7783 C.code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7784 C.value_forbidden:n = true ,
7785 t.code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7786 t.value_forbidden:n = true ,
7787 T.code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7788 T.value_forbidden:n = true ,
7789 b.code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7790 b.value_forbidden:n = true ,
7791 B.code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7792 B.value_forbidden:n = true ,
7793 m.code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7794 m.value_forbidden:n = true ,
7795 v-center.meta:n = m ,
7796 p.code:n = \bool_set_true:N \l_@@_p_block_bool ,
7797 p.value_forbidden:n = true ,
7798 respect-arraystretch.code:n =
7799 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7800 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.

```

7801 color.code:n =
7802 \@@_color:n { #1 }
7803 \tl_set_rescan:Nnn
7804 \l_@@_draw_tl
7805 { \char_set_catcode_other:N ! }
7806 { #1 } ,
7807 color.value_required:n = true ,
7808 }

```

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.

```

7809 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7810 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7811 {

```

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).

```

7812 \tl_if_blank:nTF { #2 }
7813 { \@@_Block_ii:nnnnn 1 1 }
7814 {
7815 \tl_if_in:nnTF { #2 } { - }
7816 {
7817 \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7818 \@@_Block_i_czech:w \@@_Block_i:w
7819 #2 \q_stop
7820 }
7821 {
7822 \@@_error:nn { Bad~argument~for~Block } { #2 }
7823 \@@_Block_ii:nnnnn 1 1
7824 }
7825 }
7826 { #1 } { #3 } { #4 }
7827 \ignorespaces
7828 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7829 \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`.

```
7830 {
7831   \char_set_catcode_active:N -
7832   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7833 }
```

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.

```
7834 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7835 {
```

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).

```
7836   \bool_lazy_or:nnTF
7837     { \tl_if_blank_p:n { #1 } }
7838     { \str_if_eq_p:ee { * } { #1 } }
7839     { \int_set:Nn \l_tmpa_int { 100 } }
7840     { \int_set:Nn \l_tmpa_int { #1 } }
7841   \bool_lazy_or:nnTF
7842     { \tl_if_blank_p:n { #2 } }
7843     { \str_if_eq_p:ee { * } { #2 } }
7844     { \int_set:Nn \l_tmpb_int { 100 } }
7845     { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7846   \int_compare:nNnTF \l_tmpb_int = 1
7847   {
7848     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7849     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7850     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7851   }
7852   { \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.

```
7853   \keys_set:known:n { nicematrix / BlockFirstPass } { #3 }
7854   \tl_set:Ne \l_tmpa_tl
7855   {
7856     { \int_use:N \c@iRow }
7857     { \int_use:N \c@jCol }
7858     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7859     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7860   }
```

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).

```
7861   \bool_set_false:N \l_tmpa_bool
7862   \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7863     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7864   \bool_case:nF
7865   {
7866     \l_tmpa_bool                { \@@_Block_vii:eenenn }
7867     \l_@@_p_block_bool          { \@@_Block_vi:eenenn }

```

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.

```

7868     \l_@@_X_bool                { \@@_Block_v:eenenn }
7869     { \tl_if_empty_p:n { #5 } }  { \@@_Block_v:eenenn }
7870     { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
7871     { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
7872   }
7873   { \@@_Block_v:eenenn }
7874   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7875 }

```

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.

```

7876 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7877 {
7878   \int_gincr:N \g_@@_block_box_int
7879   \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7880   \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7881   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7882   {
7883     \tl_gput_right:Ne \g_@@_rules_tl
7884     {
7885       \@@_draw_diagbox:nnnnnn
7886       { \int_use:N \c@iRow }
7887       { \int_use:N \c@jCol }
7888       { \int_eval:n { \c@iRow + #1 - 1 } }
7889       { \int_eval:n { \c@jCol + #2 - 1 } }
7890       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7891       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7892     }
7893   }
7894   \box_gclear_new:c
7895   { g_@@_block_box _ \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!

```

7896   \hbox_gset:cn
7897   { g_@@_block_box _ \int_use:N \g_@@_block_box_int _ box }
7898   {

```

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`).

```

7899      \tl_if_empty:NTF \l_@@_color_tl
7900      { \int_compare:nNnT { #2 } = 1 \set@color }
7901      { \@@_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`.

```

7902      \int_compare:nNnT { #1 } = 1
7903      {
7904          \int_if_zero:nTF \c@iRow
7905          {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7906      \cs_set_eq:NN \Block \@@_NullBlock:
7907      \l_@@_code_for_first_row_tl
7908      }
7909      {
7910          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7911          {
7912              \cs_set_eq:NN \Block \@@_NullBlock:
7913              \l_@@_code_for_last_row_tl
7914          }
7915      }
7916      \g_@@_row_style_tl
7917      }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7918      \@@_reset_arraystretch:
7919      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7920      #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.).

```

7921      \@@_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}`.

```

7922 \bool_if:NTF \l_@@_tabular_bool
7923 {
7924   \bool_lazy_all:nTF
7925   {
7926     { \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.

```

7927     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7928     { ! \g_@@_rotate_bool }
7929   }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7930   {
7931     \use:e
7932     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7933       \exp_not:N \begin { minipage }
7934       [ \str_lowercase:f \l_@@_vpos_block_str ]
7935       { \l_@@_col_width_dim }
7936       \str_case:on \l_@@_hpos_block_str
7937       { c \centering r \raggedleft l \raggedright }
7938     }
7939     #5
7940   \end { minipage }
7941 }

```

In the other cases, we use a `{tabular}`.

```

7942   {
7943     \use:e
7944     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7945       \exp_not:N \begin { tabular }
7946       [ \str_lowercase:f \l_@@_vpos_block_str ]
7947       { @ { } \l_@@_hpos_block_str @ { } }
7948     }
7949     #5
7950   \end { tabular }
7951 }
7952 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7953   {
7954     $ % $
7955     \bool_if:NT \l_@@_small_bool
7956     {
7957       \def \arraystretch { 0.47 }
7958       \dim_set:Nn \arraycolsep { 1.45 pt }
7959     }
7960     \use:e
7961     {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7962       \exp_not:N \begin { array }
7963       [ \str_lowercase:f \l_@@_vpos_block_str ]
7964       { @ { } \l_@@_hpos_block_str @ { } }
7965     }
7966     \@@_tuning_key_small:
7967     #5
7968   \end { array }
7969   $ % $
7970 }
7971 }

```

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).

```
7972 \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.

```
7973 \int_compare:nNnT { #2 } = 1
7974 {
7975   \dim_gset:Nn \g_@@_blocks_wd_dim
7976   {
7977     \dim_max:nn
7978     \g_@@_blocks_wd_dim
7979     {
7980       \box_wd:c
7981       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7982     }
7983   }
7984 }
```

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.

```
7985 \int_compare:nNnT { #1 } = 1
7986 {
7987   \bool_lazy_any:nT
7988   {
7989     { \str_if_empty_p:N \l_@@_vpos_block_str }
7990     { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7991     { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7992   }
7993   \@@_adjust_blocks_ht_dp:
7994 }
7995 \seq_gput_right:Ne \g_@@_blocks_seq
7996 {
7997   \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.

```
7998 {
7999   \exp_not:n { #3 } ,
8000   \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
8001 \bool_if:NT \g_@@_rotate_bool
8002 {
8003   \bool_if:NTF \g_@@_rotate_c_bool
8004   { m }
8005   {
8006     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8007     { T }
8008   }
8009 }
8010 }
8011 {
8012   \box_use_drop:c
8013   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8014 }
8015 }
8016 \bool_set_false:N \g_@@_rotate_c_bool
8017 }
```

```

8018 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
8019 {
8020   \dim_gset:Nn \g_@@_blocks_ht_dim
8021   {
8022     \dim_max:nn
8023     \g_@@_blocks_ht_dim
8024     {
8025       \box_ht:c
8026       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8027     }
8028   }
8029   \dim_gset:Nn \g_@@_blocks_dp_dim
8030   {
8031     \dim_max:nn
8032     \g_@@_blocks_dp_dim
8033     {
8034       \box_dp:c
8035       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8036     }
8037   }
8038 }

8039 \cs_new:Npn \@@_adjust_hpos_rotate:
8040 {
8041   \bool_if:NT \g_@@_rotate_bool
8042   {
8043     \str_set:Ne \l_@@_hpos_block_str
8044     {
8045       \bool_if:NTF \g_@@_rotate_c_bool
8046       c
8047       {
8048         \str_case:onF \l_@@_vpos_block_str
8049         { b l B l t r T r }
8050         {
8051           \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
8052           r
8053           l
8054         }
8055       }
8056     }
8057   }
8058 }
8059 \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*.

```

8060 \cs_new_protected:Npn \@@_rotate_box_of_block:
8061 {
8062   \box_grotate:cn
8063   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8064   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8065   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8066   {
8067     \vbox_gset_top:cn
8068     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8069     {
8070       \skip_vertical:n { 0.8 ex }
8071       \box_use:c
8072       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8073     }
8074   }
8075   \bool_if:NT \g_@@_rotate_c_bool

```

```

8076 {
8077   \hbox_gset:cn
8078   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8079   {
8080     \IfFormatAtLeastTF { 2026-04-01 }
8081     {
8082       \vbox_center:n
8083       {
8084         \box_use:c
8085         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8086       }
8087     }
8088     {
8089       $ % $
8090       \vcenter
8091       {
8092         \box_use:c
8093         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8094       }
8095       $ % $
8096     }
8097   }
8098 }
8099 }

```

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 *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.

```

8100 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8101 {
8102   \seq_gput_right:Ne \g_@@_blocks_seq
8103   {
8104     \l_tmpa_tl
8105     { \exp_not:n { #3 } }
8106     {
8107       \bool_if:NTF \l_@@_tabular_bool
8108       {
8109         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8110 \@@_reset_arraystretch:
8111 \exp_not:n
8112 {
8113   \dim_zero:N \extrarowheight
8114   #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).

```

8115 \tag_if_active:T { \tag_stop:n { table } }
8116 \use:e
8117 {
8118   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8119   { @ { } \l_@@_hpos_block_str @ { } }
8120 }
8121 #5
8122 \end { tabular }

```

```

8123     }
8124   }
8125 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

8126   {
8127   {

```

The following will be no-op when `respect-arraystretch` is in force.

```

8128     \@@_reset_arraystretch:
8129     \exp_not:n
8130     {
8131       \dim_zero:N \extrarowheight
8132       #4
8133       $ % $
8134       \use:e
8135       {
8136         \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8137         { @ { } \l_@@_hpos_block_str @ { } }
8138       }
8139       \@@_tuning_key_small:
8140       #5
8141       \end { array }
8142       $ % $
8143     }
8144   }
8145 }
8146 }
8147 }
8148 }
8149 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8150 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8151 {
8152   \seq_gput_right:Ne \g_@@_blocks_seq
8153   {
8154     \l_tmpa_tl
8155     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8156     { { \exp_not:n { #4 #5 } } }
8157   }
8158 }
8159 \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`.

```

8160 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8161 {
8162   \seq_gput_right:Ne \g_@@_blocks_seq
8163   {
8164     \l_tmpa_tl
8165     { \exp_not:n { #3 } }
8166     { \exp_not:n { #4 #5 } }
8167   }
8168 }
8169 \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).

```

8170 \keys_define:nn { nicematrix / Block }

```

```

8171 {
8172     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8173     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8174 tikz .code:n =
8175     \IfPackageLoadedTF { tikz }
8176     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8177     { \@@_error:n { tikz~key~without~tikz } } ,
8178 tikz .value_required:n = true ,
8179 fill .code:n =
8180     \tl_set_rescan:Nnn
8181     \l_@@_fill_tl
8182     { \char_set_catcode_other:N ! }
8183     { #1 } ,
8184 fill .value_required:n = true ,

```

In fine, the opacity will be applied by `\pgfsetfillopacity`.

```

8185 opacity .tl_set:N = \l_@@_opacity_tl ,
8186 opacity .value_required:n = true ,
8187 draw .code:n =
8188     \tl_set_rescan:Nnn
8189     \l_@@_draw_tl
8190     { \char_set_catcode_other:N ! }
8191     { #1 } ,
8192 draw .default:n = default ,
8193 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8194 rounded-corners .default:n = 4 pt ,
8195 color .code:n =
8196     \@@_color:n { #1 }
8197     \tl_set_rescan:Nnn
8198     \l_@@_draw_tl
8199     { \char_set_catcode_other:N ! }
8200     { #1 } ,
8201 borders .clist_set:N = \l_@@_borders_clist ,
8202 borders .default:n = all ,
8203 hvlines .meta:n = { vlines , hlines } ,
8204 vlines .bool_set:N = \l_@@_vlines_block_bool ,
8205 hlines .bool_set:N = \l_@@_hlines_block_bool ,
8206 rules/width .dim_set:N = \arrayrulewidth ,
8207 rules/color .tl_set:N = \l_@@_rules_color_tl ,
8208 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8209 line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8210 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8211     \bool_set_true:N \l_@@_p_block_bool ,
8212 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8213 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8214 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8215 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8216     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8217 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8218     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8219 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8220     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8221 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8222 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8223 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8224 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8225 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8226 m .value_forbidden:n = true ,
8227 v-center .meta:n = m ,

```

```

8228 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8229 p .value_forbidden:n = true ,
8230 name .str_set:N = \l_@@_block_name_str ,
8231 name .value_required:n = true ,
8232 respect-arraystretch .code:n =
8233   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8234 respect-arraystretch .value_forbidden:n = true ,
8235 transparent .bool_set:N = \l_@@_transparent_bool ,
8236 unknown .code:n =
8237   \@@_unknown_key:nn
8238     { nicematrix / Block }
8239     { Unknown-key-for-Block }
8240 }

```

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.

```

8241 \cs_new_protected:Npn \@@_draw_blocks:
8242 {
8243   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8244   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8245 }
8246 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8247 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8248 \int_zero:N \l_@@_last_row_int
8249 \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”).

```

8250 \int_compare:nNnTF { #3 } > { 98 }
8251 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8252 { \int_set:Nn \l_@@_last_row_int { #3 } }
8253 \int_compare:nNnTF { #4 } > { 98 }
8254 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8255 { \int_set:Nn \l_@@_last_col_int { #4 } }
8256 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8257 {
8258   \bool_lazy_and:nnTF
8259     \l_@@_preamble_bool
8260     {
8261       \int_compare_p:n
8262         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8263     }
8264     {
8265       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8266       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8267       \@@_msg_redirect_name:nn { columns-not-used } { none }
8268     }
8269     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8270 }
8271 {
8272   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8273   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8274   {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8275         \use:e
8276         {
8277             \@@_Block_v:nnnnnn
8278             { #1 }
8279             { #2 }
8280             { \int_use:N \l_@@_last_row_int }
8281             { \int_use:N \l_@@_last_col_int }
8282         }
8283         { #5 }
8284         { #6 }
8285     }
8286 }
8287 }

```

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.

```

8288 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8289 {

```

The group is for the keys.

```

8290     \group_begin:
8291     \int_compare:nNtT { #1 } = { #3 }
8292     { \str_set:Nn \l_@@_vpos_block_str { t } }
8293     \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`.

```

8294     \bool_if:NT \l_@@_amp_in_blocks_bool
8295     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }
8296     \bool_lazy_and:nnT
8297     \l_@@_vlines_block_bool
8298     { ! \l_@@_ampersand_bool }
8299     {
8300         \tl_gput_right:Ne \g_@@_rules_tl
8301         {
8302             \@@_vlines_block:nnnnnn
8303             { \dim_use:N \arrayrulewidth }
8304             { \l_@@_rules_color_tl }
8305             { #1 } { #2 } { #3 } { #4 }
8306         }
8307     }
8308     \bool_if:NT \l_@@_hlines_block_bool
8309     {
8310         \tl_gput_right:Ne \g_@@_rules_tl
8311         {
8312             \@@_hlines_block:nnnnnn
8313             { \dim_use:N \arrayrulewidth }
8314             { \l_@@_rules_color_tl }
8315             { #1 } { #2 } { #3 } { #4 }
8316         }
8317     }

```

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).

```

8318     \bool_if:NF \l_@@_transparent_bool
8319     {
8320         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8321         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8322     }

```

```

8323 \tl_if_empty:NF \l_@@_draw_tl
8324 {
8325   \tl_gput_right:Nn \g_@@_rules_tl
8326   {
8327     \@@_stroke_block:nnnnn

```

#5 are the options

```

8328       { #5 } { #1 } { #2 } { #3 } { #4 }
8329     }
8330   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8331   { { #1 } { #2 } { #3 } { #4 } }
8332 }
8333 \clist_if_empty:NF \l_@@_borders_clist
8334 {
8335   \tl_gput_right:Nn \g_nicematrix_code_after_tl
8336   {
8337     \@@_stroke_borders_block:nnnnn
8338     { #5 } { #1 } { #2 } { #3 } { #4 }
8339   }
8340 }
8341 \tl_if_empty:NF \l_@@_fill_tl
8342 {
8343   \@@_add_opacity_to_fill:
8344   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8345   {
8346     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8347     { #1 - #2 }
8348     { #3 - #4 }
8349     { \dim_use:N \l_@@_rounded_corners_dim }
8350   }
8351 }
8352 \seq_if_empty:NF \l_@@_tikz_seq
8353 {
8354   \tl_gput_right:Ne \g_nicematrix_code_before_tl
8355   {
8356     \@@_block_tikz:nnnnn
8357     { \seq_use:Nn \l_@@_tikz_seq { , } }
8358     { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8359   }
8360 }
8361 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8362 {
8363   \tl_gput_right:Ne \g_@@_rules_tl
8364   {
8365     \@@_draw_diagbox:nnnnnn
8366     { #1 } { #2 } { #3 } { #4 }
8367     { \exp_not:n { ##1 } }
8368     { \exp_not:n { ##2 } }
8369   }
8370 }

```

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.

```

8371 \pgfpicture
8372 \pgfrememberpicturepositiononpagetrue
8373 \pgf@relevantforpicturesizefalse
8374 \@@_qpoint:n { row - #1 }
8375 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8376 \@@_qpoint:n { col - #2 }
8377 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8378 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8379 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8380 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8381 \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.

```

8382 \@@_pgf_rect_node:nnnnn
8383 { \@@_env: - #1 - #2 - block }
8384 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8385 \str_if_empty:NF \l_@@_block_name_str
8386 {
8387   \pgfnodealias
8388   { \@@_env: - \l_@@_block_name_str }
8389   { \@@_env: - #1 - #2 - block }
8390   \str_if_empty:NF \l_@@_name_str
8391   {
8392     \pgfnodealias
8393     { \l_@@_name_str - \l_@@_block_name_str }
8394     { \@@_env: - #1 - #2 - block }
8395   }
8396 }

```

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.

```

8397 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8398 {
8399   \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.

```

8400 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8401 {

```

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.

```

8402 \cs_if_exist:cT
8403 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8404 {
8405   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8406   {

```

```

8407         \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8408         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8409     }
8410 }
8411 }

```

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.

```

8412     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8413     {
8414         \@@_qpoint:n { col - #2 }
8415         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8416     }
8417     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8418     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8419     {
8420         \cs_if_exist:cT
8421         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8422         {
8423             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8424             {
8425                 \pgfpointanchor
8426                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8427                 { east }
8428                 \dim_set:Nn \l_@@_tmpd_dim
8429                 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8430             }
8431         }
8432     }
8433     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8434     {
8435         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8436         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8437     }
8438     \@@_pgf_rect_node:nnnnn
8439     { \@@_env: - #1 - #2 - block - short }
8440     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8441 }

```

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.

```

8442     \bool_if:NT \l_@@_medium_nodes_bool
8443     {
8444         \@@_pgf_rect_node:nnn
8445         { \@@_env: - #1 - #2 - block - medium }
8446         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8447         {
8448             \pgfpointanchor
8449             { \@@_env:
8450               - \int_use:N \l_@@_last_row_int
8451               - \int_use:N \l_@@_last_col_int - medium
8452             }
8453             { south~east }
8454         }
8455     }
8456     \endpgfpicture
8457

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8458     \bool_if:NTF \l_@@_ampersand_bool
8459     {
8460         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8461         \int_zero_new:N \l_@@_split_int
8462         \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).

```

8463 \int_set:Nn \l_@@_first_row_int { #1 }
8464 \int_set:Nn \l_@@_first_col_int { #2 }
8465 \int_set:Nn \l_@@_last_row_int { #3 }
8466 \int_set:Nn \l_@@_last_col_int { #4 }

8467 \pgfpicture
8468 \pgfrememberpicturepositiononpagetrue
8469 \pgf@relevantforpicturesizefalse
8470 \@@_qpoint:n { row - #1 }
8471 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8472 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8473 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8474 \@@_qpoint:n { col - #2 }
8475 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8476 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8477 \dim_set:Nn \l_tmpb_dim
8478 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8479 \bool_lazy_or:nnT
8480 \l_@@_vlines_block_bool
8481 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8482 {
8483 \int_step_inline:nn { \l_@@_split_int - 1 }
8484 {
8485 \pgfpathmoveto
8486 {
8487 \pgfpoint
8488 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8489 \l_@@_tmpc_dim
8490 }
8491 \pgfpathlineto
8492 {
8493 \pgfpoint
8494 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8495 \l_@@_tmpd_dim
8496 }
8497 \CT@arc@
8498 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8499 \pgfsetrectcap
8500 \pgfusepathqstroke
8501 }
8502 }
8503 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8504 \int_zero_new:N \l_@@_split_i_int

8505 \str_if_eq:eeTF \l_@@_vpos_block_str T
8506 {
8507 \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8508 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8509 }
8510 {
8511 \str_if_eq:eeTF \l_@@_vpos_block_str B
8512 {
8513 \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8514 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8515 }
8516 {
8517 \bool_lazy_or:nnTF
8518 { \int_compare_p:nNn { #1 } = { #3 } }
8519 { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8520 {
8521 \@@_qpoint:n { row - #1 - base }

```

```

8522         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8523     }
8524     {
8525         \str_if_eq:eeTF \l_@@_vpos_block_str b
8526         {
8527             \@@_qpoint:n { row - #3 - base }
8528             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8529         }
8530         {
8531             \@@_qpoint:n { #1 - #2 - block }
8532             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8533         }
8534     }
8535 }
8536 }
8537 \int_step_inline:nn \l_@@_split_int
8538 {
8539     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8540     \int_set:Nn \l_@@_split_i_int { ##1 }
8541     \dim_set:Nn \col@sep
8542     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8543     \pgftransformshift
8544     {
8545         \pgfpoint
8546         {
8547             \l_tmpa_dim + ##1 \l_tmpb_dim -
8548             \str_case:on \l_@@_hpos_block_str
8549             {
8550                 l { \l_tmpb_dim + \col@sep }
8551                 c { 0.5 \l_tmpb_dim }
8552                 r { \col@sep }
8553             }
8554         }
8555         { \l_@@_tmpc_dim }
8556     }
8557     \pgfset { inner~sep = \c_zero_dim }
8558     \pgfnode
8559     { rectangle }
8560     {
8561         \str_if_eq:eeTF T \l_@@_vpos_block_str
8562         {
8563             \str_case:on \l_@@_hpos_block_str
8564             {
8565                 l { north~west }
8566                 c { north }
8567                 r { north~east }
8568             }
8569         }
8570         {
8571             \str_if_eq:eeTF B \l_@@_vpos_block_str
8572             {
8573                 \str_case:on \l_@@_hpos_block_str
8574                 {
8575                     l { south~west }
8576                     c { south }
8577                     r { south~east }
8578                 }
8579             }
8580             {
8581                 \bool_lazy_any:nTF
8582                 {
8583                     { \int_compare_p:nNn { #1 } = { #3 } }

```

```

8584         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8585         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8586     }
8587     {
8588         \str_case:on \l_@@_hpos_block_str
8589         {
8590             l { base~west }
8591             c { base }
8592             r { base~east }
8593         }
8594     }
8595     {
8596         \str_case:on \l_@@_hpos_block_str
8597         {
8598             l { west }
8599             c { center }
8600             r { east }
8601         }
8602     }
8603 }
8604 }
8605 }
8606 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8607 \group_end:
8608 }
8609 \endpgfpicture
8610 }

```

Now the case where there is no ampersand & in the content of the block.

```

8611 {
8612     \bool_if:NTF \l_@@_p_block_bool
8613     {

```

When the final user has used the key p, we have to compute the width.

```

8614     \pgfpicture
8615     \pgfrememberpicturepositiononpagetrue
8616     \pgf@relevantforpicturesizefalse
8617     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8618     {
8619         \@@_qpoint:n { col - #2 }
8620         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8621         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8622     }
8623     {
8624         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8625         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8626         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8627     }
8628     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8629 \endpgfpicture
8630 \hbox_set:Nn \l_@@_cell_box
8631 {
8632     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8633     { \g_tmpb_dim }
8634     \str_case:on \l_@@_hpos_block_str
8635     { c \centering r \raggedleft l \raggedright j { } }
8636     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8637     #6
8638     \end { minipage }
8639 }
8640 }
8641 {
8642     \hbox_set:Nn \l_@@_cell_box
8643     {

```

```

8644         \set@color
8645         \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8646         #6
8647     }
8648 }
8649 \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.

```

8650 \pgfpicture
8651 \pgfrememberpicturepositiononpagetrue
8652 \pgf@relevantforpicturesizefalse
8653 \bool_lazy_any:nTF
8654 {
8655     { \str_if_empty_p:N \l_@@_vpos_block_str }
8656     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8657     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8658     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8659 }
8660 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8661     \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`.

```

8662     \bool_if:nT \g_@@_last_col_found_bool
8663     {
8664         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8665         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8666     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8667     \tl_set:Ne \l_tmpa_tl
8668     {
8669         \str_case:on \l_@@_vpos_block_str
8670         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8671         { } {
8672             \str_case:on \l_@@_hpos_block_str
8673             {
8674                 c { center }
8675                 l { west }
8676                 r { east }
8677                 j { center }
8678             }
8679         }
8680         c {
8681             \str_case:on \l_@@_hpos_block_str
8682             {
8683                 c { center }
8684                 l { west }
8685                 r { east }
8686                 j { center }
8687             }
8688         }
8689     }
8690     T {
8691         \str_case:on \l_@@_hpos_block_str
8692         {
8693             c { north }
8694             l { north-west }

```

```

8695             r { north-east }
8696             j { north }
8697         }
8698
8699     }
8700     B {
8701         \str_case:on \l_@@_hpos_block_str
8702         {
8703             c { south }
8704             l { south-west }
8705             r { south-east }
8706             j { south }
8707         }
8708
8709     }
8710 }
8711
8712 \pgftransformshift
8713 {
8714     \pgfpointanchor
8715     {
8716         \@@_env: - #1 - #2 - block
8717         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8718     }
8719     \l_tmpa_tl
8720 }
8721 \pgfset { inner~sep = \c_zero_dim }
8722 \pgfnode
8723 { rectangle }
8724 \l_tmpa_tl
8725 { \box_use_drop:N \l_@@_cell_box } { } { }
8726 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8727 {
8728     \pgfextracty \l_tmpa_dim
8729     {
8730         \@@_qpoint:n
8731         {
8732             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8733             - base
8734         }
8735     }
8736     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8737 \pgfpointanchor
8738 {
8739     \@@_env: - #1 - #2 - block
8740     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8741 }
8742 {
8743     \str_case:on \l_@@_hpos_block_str
8744     {
8745         c { center }
8746         l { west }
8747         r { east }
8748         j { center }
8749     }
8750 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8751 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8752 \pgfset { inner~sep = \c_zero_dim }

```

```

8753     \pgfnode
8754     { rectangle }
8755     {
8756       \str_case:on \l_@@_hpos_block_str
8757       {
8758         c { base }
8759         l { base~west }
8760         r { base~east }
8761         j { base }
8762       }
8763     }
8764     { \box_use_drop:N \l_@@_cell_box } { } { }
8765   }
8766   \endpgfpicture
8767 }
8768 \group_end:
8769 }
8770 \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).

```

8771 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8772 {
8773   \tl_if_empty:NF \l_@@_opacity_tl
8774   {
8775     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8776     {
8777       \tl_set:Nc \l_@@_fill_tl
8778       {
8779         [ opacity = \l_@@_opacity_tl ,
8780         \tl_tail:o \l_@@_fill_tl
8781       }
8782     }
8783     {
8784       \tl_set:Nc \l_@@_fill_tl
8785       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8786     }
8787   }
8788 }

```

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*.

```

8789 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8790 {
8791   \group_begin:
8792   \tl_clear:N \l_@@_draw_tl
8793   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8794   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8795   \tl_if_empty:NF \l_@@_draw_tl
8796   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8797     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8798     \CT@arc@
8799     { \@@_color:o \l_@@_draw_tl }
8800   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8801   \pgfsetcornersarced
8802   { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }

```

```

8803 \int_compare:nNf { #2 } > \c@iRow
8804 {
8805   \int_compare:nNf { #3 } > \c@jCol
8806   {
8807     \@@_qpoint:n { row - #2 }
8808     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8809     \@@_qpoint:n { col - #3 }
8810     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8811     \@@_qpoint:n
8812     { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8813     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8814     \@@_qpoint:n
8815     { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8816     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8817     \pgfpathrectanglecorners
8818     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8819     { \pgfpoint \pgf@x \l_tmpa_dim }
8820     \dim_compare:nNf \l_@@_rounded_corners_dim = \c_zero_dim
8821     \pgfusepathqstroke
8822     { \pgfusepath { stroke } }
8823   }
8824 }
8825 \group_end:
8826 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8827 \keys_define:nn { nicematrix / BlockStroke }
8828 {
8829   color .tl_set:N = \l_@@_draw_tl ,
8830   draw .code:n =
8831     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8832   draw .default:n = default ,
8833   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8834   rounded-corners .default:n = 4 pt ,
8835   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8836   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8837 }

```

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*.

```

8838 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8839 {
8840   \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).

```

8841   \dim_set:Nn \arrayrulewidth { #1 }
8842   \pgfsetlinewidth \arrayrulewidth
8843   \@@_set_CTarc:n { #2 }
8844   \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).

```

8845   \seq_set_filter:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8846   {
8847     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8848     ||
8849     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }

```

```

8850     ||
8851     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8852     ||
8853     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8854 }
8855 \bool_if:NT \l_@@_fix_vertex_bool
8856 {
8857     \int_compare:nNnT { #4 } > 1
8858     {
8859         \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8860         {
8861             { 1 }
8862             { 1 }
8863             { \int_use:N \c@iRow }
8864             { \int_eval:n { #4 -1 } }
8865             { }
8866         }
8867     }
8868     \int_compare:nNnT { #6 } < \c@jCol
8869     {
8870         \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8871         {
8872             { 1 }
8873             { \int_eval:n { #6 + 1 } }
8874             { \int_use:N \c@iRow }
8875             { \int_use:N \c@jCol }
8876             { }
8877         }
8878     }
8879 }
8880 \int_set:Nn \l_@@_start_int { #3 }
8881 \int_set:Nn \l_@@_end_int { #5 }
8882 \int_step_inline:nnn { #4 } { #6 + 1 }
8883 {
8884     \int_set:Nn \l_@@_position_int { ##1 }
8885     \socket_use:n { nicematrix / draw-vrule }
8886 }
8887 \group_end:
8888 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8889 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8890 {
8891     \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).

```

8892     \dim_set:Nn \arrayrulewidth { #1 }
8893     \pgfsetlinewidth \arrayrulewidth
8894     \@@_set_CTarc:n { #2 }
8895     \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).

```

8896     \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8897     {
8898         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8899         ||
8900         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }

```

```

8901      ||
8902      \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8903      ||
8904      \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8905    }
8906    \bool_if:NT \l_@@_fix_vertex_bool
8907    {
8908      \int_compare:nNnT { #3 } > 1
8909      {
8910        \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8911        {
8912          { 1 }
8913          { 1 }
8914          { \int_eval:n { #3 - 1 } }
8915          { \int_use:N \c@jCol }
8916          { }
8917        }
8918      }
8919      \int_compare:nNnT { #5 } < \c@iRow
8920      {
8921        \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8922        {
8923          { \int_eval:n { #5 + 1 } }
8924          { 1 }
8925          { \int_use:N \c@iRow }
8926          { \int_use:N \c@jCol }
8927          { }
8928        }
8929      }
8930    }
8931    \int_set:Nn \l_@@_start_int { #4 }
8932    \int_set:Nn \l_@@_end_int { #6 }
8933    \int_step_inline:nnn { #3 } { #5 + 1 }
8934    {
8935      \int_set:Nn \l_@@_position_int { ##1 }
8936      \socket_use:n { nicematrix / draw-hrule }
8937    }
8938    \group_end:
8939  }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8940 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8941 {
8942   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8943   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8944
8945   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8946   { \@@_error:n { borders~forbidden } }
8947   {
8948     \tl_clear_new:N \l_@@_borders_tikz_tl
8949     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8950     \tl_set:Nn \l_@@_tmpc_tl { #2 }
8951     \tl_set:Nn \l_@@_tmpd_tl { #3 }
8952     \tl_set:Nn \l_tmpa_tl { \int_eval:n { #4 + 1 } }
8953     \tl_set:Nn \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8954     \@@_stroke_borders_block_i:
8955   }
8956 }
8957 \AtBeginDocument
8958 {

```

```

8959 \cs_new_protected:Npe \@@_stroke_borders_block_i:
8960 {
8961     \c_@@_pgfortikzpicture_tl
8962     \@@_stroke_borders_block_ii:
8963     \c_@@_endpgfortikzpicture_tl
8964 }
8965 }

8966 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8967 {
8968     \pgfrememberpicturepositiononpagetrue
8969     \pgf@relevantforpicturesizefalse
8970     \CT@arc@
8971     \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).

```

8972 \bool_set_false:N \l_tmpa_bool
8973 \clist_if_in:NnT \l_@@_borders_clist { right }
8974 {
8975     \@@_stroke_vertical:n \l_tmpb_tl
8976     \bool_set_true:N \l_tmpa_bool
8977 }
8978 \clist_if_in:NnT \l_@@_borders_clist { left }
8979 {
8980     \@@_stroke_vertical:n \l_@@_tmpd_tl
8981     \bool_set_true:N \l_tmpa_bool
8982 }
8983 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8984 {
8985     \@@_stroke_horizontal:n \l_tmpa_tl
8986     \bool_set_true:N \l_tmpa_bool
8987 }
8988 \clist_if_in:NnT \l_@@_borders_clist { top }
8989 {
8990     \@@_stroke_horizontal:n \l_@@_tmpc_tl
8991     \bool_set_true:N \l_tmpa_bool
8992 }
8993 \bool_if:NF \l_tmpa_bool
8994 {
8995     \@@_stroke_vertical:n \l_tmpb_tl
8996     \@@_stroke_vertical:n \l_@@_tmpd_tl
8997     \@@_stroke_horizontal:n \l_tmpa_tl
8998     \@@_stroke_horizontal:n \l_@@_tmpc_tl
8999 }
9000 }

9001 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
9002 {
9003     tikz .code:n =
9004         \cs_if_exist:NTF \tikzpicture
9005             { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
9006             { \@@_error:n { tikz~in~borders~without~tikz } } ,
9007     tikz .value_required:n = true ,
9008     dashed .meta:n = { tikz = dashed } ,
9009     top .code:n = ,
9010     bottom .code:n = ,
9011     left .code:n = ,
9012     right .code:n = ,
9013     all .code:n = ,
9014     unknown .code:n = \@@_error:n { bad~border }
9015 }

```

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).

```

9016 \cs_new_protected:Npn \@@_stroke_vertical:n #1
9017 {
9018   \@@_qpoint:n \l_@@_tmpc_tl
9019   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9020   \@@_qpoint:n \l_tmpa_tl
9021   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9022   \@@_qpoint:n { #1 }
9023   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9024   {
9025     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
9026     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
9027     \pgfusepathqstroke
9028   }
9029   {
9030     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9031     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
9032   }
9033 }

```

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).

```

9034 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
9035 {
9036   \@@_qpoint:n \l_@@_tmpd_tl
9037   \clist_if_in:NnTF \l_@@_borders_clist { left }
9038   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9039   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9040   \@@_qpoint:n \l_tmpb_tl
9041   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9042   \@@_qpoint:n { #1 }
9043   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9044   {
9045     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9046     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9047     \pgfusepathqstroke
9048   }
9049   {
9050     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9051     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9052   }
9053 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

9054 \keys_define:nn { nicematrix / BlockBorders }
9055 {
9056   borders .clist_set:N = \l_@@_borders_clist ,
9057   borders .default:n = all ,
9058   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9059   rounded-corners .default:n = 4 pt ,
9060   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9061   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9062 }

```

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.

```

9063 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9064 {
9065   \begin { tikzpicture }
9066   \@@_clip_with_rounded_corners:

```

We use `\clist_map_inline:nn` because `#5` is a list of lists.

```

9067   \clist_map_inline:nn { #1 }
9068   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9069     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9070     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9071     (
9072       [
9073         xshift = \dim_use:N \l_@@_xoffset_dim ,
9074         yshift = - \dim_use:N \l_@@_yoffset_dim
9075       ]
9076       #2 -| #3
9077     )
9078     rectangle
9079     (
9080       [
9081         xshift = - \dim_use:N \l_@@_xoffset_dim ,
9082         yshift = \dim_use:N \l_@@_yoffset_dim
9083       ]
9084       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9085     ) ;
9086   }
9087   \end { tikzpicture }
9088 }
9089 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

9090 \keys_define:nn { nicematrix / SpecialOffset }
9091 {
9092   xoffset .dim_set:N = \l_@@_xoffset_dim ,
9093   yoffset .dim_set:N = \l_@@_yoffset_dim ,
9094   offset .meta:n = { xoffset = #1 , yoffset = #1 }
9095 }

```

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.

```

9096 \cs_new_protected:Npn \@@_NullBlock:
9097 { \@@_collect_options:n { \@@_NullBlock_i: } }
9098 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9099 { }

```

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.

```

9100 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9101 {
9102   \tl_gput_right:Ne \g_@@_pre_code_before_tl
9103   {

```

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`).

```

9104     \@@_subcellcolor:nnnnnnn
9105     {
9106       \tl_if_blank:nTF { #1 }
9107       { { \exp_not:n { #2 } } }
9108       { [ #1 ] { \exp_not:n { #2 } } }
9109     }
9110     { \int_use:N \l_@@_first_row_int } % first row of the block

```

```

9111         { \int_use:N \l_@@_first_col_int } % first column of the block
9112         { \int_use:N \l_@@_last_row_int } % last row of the block
9113         { \int_use:N \l_@@_last_col_int } % last column of the block
9114         { \int_use:N \l_@@_split_int }
9115         { \int_use:N \l_@@_split_i_int }
9116     }
9117     \ignorespaces
9118 }
9119 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9120 {
9121     \@@_color_opacity: #1
9122     \pgfpicture
9123     \pgf@relevantforpicturesizefalse
9124     \@@_qpoint:n { col - #3 }
9125     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9126     \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9127     \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9128     \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9129     \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9130     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9131     \@@_qpoint:n { row - #2 }
9132     \pgfpathrectanglecorners
9133         { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9134         { \pgfpoint \l_tmpb_dim \pgf@y }
9135     \pgfusepathqfill
9136     \endpgfpicture
9137 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

9138 \keys_define:nn { nicematrix / Auto }
9139 {
9140     columns-type .tl_set:N = \l_@@_columns_type_tl ,
9141     columns-type .value_required:n = true ,
9142     l .meta:n = { columns-type = l } ,
9143     r .meta:n = { columns-type = r } ,
9144     c .meta:n = { columns-type = c } ,
9145     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9146     delimiters / color .value_required:n = true ,
9147     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9148     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9149     delimiters .value_required:n = true ,
9150     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9151     rounded-corners .default:n = 4 pt
9152 }
9153 \NewDocumentCommand \AutoNiceMatrixWithDelims
9154 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9155 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
9156 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9157 {

```

The group is for the protection of the keys.

```

9158     \group_begin:
9159     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9160     \use:e
9161     {
9162         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }

```

```

9163         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9164         [ \exp_not:o \l_tmpa_tl ]
9165     }
9166     \int_if_zero:nT \l_@@_first_row_int
9167     {
9168         \int_if_zero:nT \l_@@_first_col_int { & }
9169         \prg_replicate:nn { #4 - 1 } { & }
9170         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9171     }
9172     \prg_replicate:nn { #3 }
9173     {
9174         \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).

```

9175         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9176         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9177     }
9178     \int_compare:nNnT \l_@@_last_row_int > { -2 }
9179     {
9180         \int_if_zero:nT \l_@@_first_col_int { & }
9181         \prg_replicate:nn { #4 - 1 } { & }
9182         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9183     }
9184     \end { NiceArrayWithDelims }
9185     \group_end:
9186 }
9187 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9188 {
9189     \cs_set_protected:cpn { #1 AutoNiceMatrix }
9190     {
9191         \bool_gset_true:N \g_@@_delims_bool
9192         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9193         \AutoNiceMatrixWithDelims { #2 } { #3 }
9194     }
9195 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

9196 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
9197 {
9198     \group_begin:
9199     \bool_gset_false:N \g_@@_delims_bool
9200     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9201     \group_end:
9202 }

```

28 The redefinition of the command \dotfill

```

9203 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9204 \cs_new_protected:Npn \@@_dotfill:
9205 {

```

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}).

```

9206     \@@_old_dotfill:
9207     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9208 }

```

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 its 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`.

```

9209 \cs_new_protected:Npn \@@_dotfill_i:
9210 {
9211   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9212   { \@@_old_dotfill: }
9213 }

```

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.

```

9214 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9215 {
9216   \tl_gput_right:Ne \g_@@_rules_tl
9217   {
9218     \@@_draw_diagbox:nnnnnn
9219     { \int_use:N \c@iRow }
9220     { \int_use:N \c@jCol }
9221     { \int_use:N \c@iRow }
9222     { \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.

```

9223     { \g_@@_row_style_tl \exp_not:n { #1 } }
9224     { \g_@@_row_style_tl \exp_not:n { #2 } }
9225   }

```

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`.

```

9226   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9227   {
9228     { \int_use:N \c@iRow }
9229     { \int_use:N \c@jCol }
9230     { \int_use:N \c@iRow }
9231     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9232     { }
9233   }
9234 }

```

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.

```

9235 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9236 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9237   \begin { pgfscope }
9238     \@@_qpoint:n { row - #1 }
9239     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9240     \@@_qpoint:n { col - #2 }
9241     \dim_set_eq:NN \l_tmpb_dim \pgf@x
9242     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9243     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9244     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9245     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9246     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9247     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9248     {

```

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.

```

9249     \CT@arc@
9250     \pgfsetroundcap
9251     \pgfusepathqstroke
9252   }
9253   \pgfset { inner~sep = 1 pt }
9254   \pgfscope
9255   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9256   \pgfnode { rectangle } { south~west }
9257   {
9258     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9259     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9260     \end { minipage }
9261   }
9262   { }
9263   { }
9264 \endpgfscope
9265 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9266 \pgfnode { rectangle } { north~east }
9267 {
9268   \begin { minipage } { 20 cm }
9269   \raggedleft
9270   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9271   \end { minipage }
9272 }
9273 { }
9274 { }
9275 \end { pgfscope }
9276 }

```

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`.

```

9277 \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 `\`.

```

9278 \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`.

```

9279 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9280 {
9281   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9282   \@@_CodeAfter_iv:n
9283 }

```

We catch the argument of the command `\end` (in #1).

```

9284 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9285 {

```

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.

```

9286   \str_if_eq:eeTF \@currenvir { #1 }
9287   { \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`.

```

9288   {
9289     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9290     \@@_CodeAfter_ii:n
9291   }
9292 }

```

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.

```

9293 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9294 {
9295   \pgfpicture
9296   \pgfrememberpicturepositiononpagetrue
9297   \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.

```

9298   \@@_qpoint:n { row - 1 }
9299   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9300   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9301   \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).

```

9302   \bool_if:nTF { #3 }
9303   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9304   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9305   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9306   {
9307     \cs_if_exist:cT
9308     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
9309     {
9310       \pgfpointanchor
9311       { \@@_env: - ##1 - #2 }
9312       { \bool_if:nTF { #3 } { west } { east } }
9313       \dim_set:Nn \l_tmpa_dim
9314       {
9315         \bool_if:nTF { #3 }
9316         {
9317           \dim_min:nn
9318           \dim_max:nn
9319           \l_tmpa_dim
9319         }

```

```

9320     }
9321   }
9322 }

```

Now we can put the delimiter with a node of PGF.

```

9323 \pgfset { inner~sep = \c_zero_dim }
9324 \dim_zero:N \nulldelimiterspace
9325 \pgftransformshift
9326 {
9327   \pgfpoint
9328   \l_tmpa_dim
9329   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9330 }
9331 \pgfnode
9332 { rectangle }
9333 { \bool_if:nTF { #3 } { east } { west } }
9334 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9335   \nullfont
9336   $ % $
9337   \@@_color:o \l_@@_delimiters_color_tl
9338   \bool_if:nTF { #3 } { \left #1 } { \left . }
9339   \vcenter
9340   {
9341     \nullfont
9342     \hrule \@height
9343       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9344       \@depth \c_zero_dim
9345       \@width \c_zero_dim
9346   }
9347   \bool_if:nTF { #3 } { \right . } { \right #1 }
9348   $ % $
9349 }
9350 { }
9351 { }
9352 \endpgfpicture
9353 }

```

32 The command \SubMatrix

```

9354 \keys_define:nn { nicematrix / sub-matrix }
9355 {
9356   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9357   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9358   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9359   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9360   xshift .value_required:n = true ,
9361   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9362   delimiters / color .value_required:n = true ,
9363   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9364   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9365   hlines .default:n = all ,
9366   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9367   vlines .default:n = all ,
9368   hvlines .meta:n = { hlines, vlines } ,
9369   hvlines .value_forbidden:n = true
9370 }
9371 \keys_define:nn { nicematrix }
9372 {
9373   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

9374 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9375 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9376 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9377 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9378 \keys_define:nn { nicematrix / SubMatrix }
9379 {
9380   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9381   delimiters / color .value_required:n = true ,
9382   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9383   hlines .default:n = all ,
9384   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9385   vlines .default:n = all ,
9386   hvlines .meta:n = { hlines, vlines } ,
9387   hvlines .value_forbidden:n = true ,
9388   name .code:n =
9389     \tl_if_empty:nTF { #1 }
9390     { \@@_error:n { Invalid-name } }
9391     {
9392       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9393       {
9394         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9395         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
9396         {
9397           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9398           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9399         }
9400       }
9401       { \@@_error:n { Invalid-name } }
9402     } ,
9403   name .value_required:n = true ,
9404   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9405   rules .value_required:n = true ,
9406   code .tl_set:N = \l_@@_code_tl ,
9407   code .value_required:n = true ,
9408   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9409 }

9410 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9411 {
9412   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9413   {
9414     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9415     [
9416       delimiters / color = \l_@@_delimiters_color_tl ,
9417       hlines = \l_@@_submatrix_hlines_clist ,
9418       vlines = \l_@@_submatrix_vlines_clist ,
9419       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9420       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9421       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9422       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9423       #5
9424     ]
9425   }
9426   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9427   \ignorespaces
9428 }

9429 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9430 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9431 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

```

```

9432 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9433 {
9434   \seq_gput_right:Ne \g_@@_submatrix_seq
9435   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9436     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9437     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9438     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9439     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9440   }
9441 }

```

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).

```

9442 \NewDocumentCommand \@@_compute_i_j:nn
9443 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9444 { \@@_compute_i_j:nnnn #1 #2 }

```

```

9445 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9446 {
9447   \def \l_@@_first_i_tl { #1 }
9448   \def \l_@@_first_j_tl { #2 }
9449   \def \l_@@_last_i_tl { #3 }
9450   \def \l_@@_last_j_tl { #4 }
9451   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9452     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9453   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9454     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9455   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9456     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9457   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9458     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9459 }

```

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`.

```

9460 \AtBeginDocument
9461 {
9462   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9463   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9464     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9465 }
9466 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9467 {
9468   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9469 \@@_compute_i_j:nn { #2 } { #3 }
9470 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
9471 { \def \arraystretch { 1 } }
9472 \bool_lazy_or:nnTF
9473 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9474 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9475 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9476 {
9477 \str_clear_new:N \l_@@_submatrix_name_str
9478 \keys_set:nn { nicematrix / SubMatrix } { #5 }
9479 \pgfpicture
9480 \pgfrememberpicturepositiononpagetrue
9481 \pgf@relevantforpicturesizefalse
9482 \pgfset { inner~sep = \c_zero_dim }
9483 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9484 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9485 \bool_if:NTF \l_@@_submatrix_slim_bool
9486 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9487 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9488 {
9489 \cs_if_exist:cT
9490 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9491 {
9492 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9493 \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9494 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9495 }
9496 \cs_if_exist:cT
9497 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9498 {
9499 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9500 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9501 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9502 }
9503 }
9504 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9505 { \@@_impossible_delimiter:n { left } }
9506 {
9507 \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9508 { \@@_impossible_delimiter:n { right } }
9509 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9510 }
9511 \endpgfpicture
9512 }
9513 \group_end:
9514 \ignorespaces
9515 }

```

The argument of the following command will be provided by curryfication.

```

9516 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9517 {
9518 \bool_if:NTF \l_@@_no_cell_nodes_bool
9519 { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9520 { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9521 }

```

`#1` is the left delimiter, `#2` is the right one, `#3` is the subscript and `#4` is the superscript.

```

9522 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9523 {

```

```

9524 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9525 \dim_set:Nn \l_@@_y_initial_dim
9526 {
9527   \fp_to_dim:n
9528   {
9529     \pgf@y
9530     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9531   }
9532 }
9533 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9534 \dim_set:Nn \l_@@_y_final_dim
9535 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9536 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9537 {
9538   \cs_if_exist:cT
9539   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9540   {
9541     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9542     \dim_set:Nn \l_@@_y_initial_dim
9543     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9544   }
9545   \cs_if_exist:cT
9546   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9547   {
9548     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9549     \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9550     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9551   }
9552 }
9553 \dim_set:Nn \l_tmpa_dim
9554 {
9555   \l_@@_y_initial_dim - \l_@@_y_final_dim +
9556   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9557 }
9558 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9559 \group_begin:
9560 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9561 \@@_set_CTarc:o \l_@@_rules_color_tl
9562 \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`.

```

9563 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9564 {
9565   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9566   {
9567     \int_compare:nNnT
9568     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9569     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9570 \@@_qpoint:n { col - ##1 }
9571 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9572 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9573 \pgfusepathqstroke
9574 }
9575 }
9576 }

```

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.

```

9577 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9578 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9579 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9580 {
9581   \bool_lazy_and:nnTF
9582   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9583   {
9584     \int_compare_p:nNn
9585     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9586   {
9587     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9588     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9589     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9590     \pgfusepathqstroke
9591   }
9592   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9593 }

```

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.

```

9594 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9595 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9596 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9597 {
9598   \bool_lazy_and:nnTF
9599   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9600   {
9601     \int_compare_p:nNn
9602     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9603   {
9604     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9605   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
9606   \dim_set:Nn \l_tmpa_dim
9607   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9608   \str_case:nn { #1 }
9609   {
9610     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9611     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9612     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9613   }
9614   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

9615   \dim_set:Nn \l_tmpb_dim
9616   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9617   \str_case:nn { #2 }
9618   {
9619     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9620     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9621     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9622   }
9623   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9624   \pgfusepathqstroke
9625   \group_end:
9626 }
9627 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9628 }

```

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).

```

9629 \str_if_empty:NF \l_@@_submatrix_name_str
9630 {
9631     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9632     \l_@@_x_initial_dim \l_@@_y_initial_dim
9633     \l_@@_x_final_dim \l_@@_y_final_dim
9634 }
9635 \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`.

```

9636 \begin { pgfscope }
9637 \pgftransformshift
9638 {
9639     \pgfpoint
9640     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9641     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9642 }
9643 \str_if_empty:NTF \l_@@_submatrix_name_str
9644 { \@@_node_left:nn #1 { } }
9645 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9646 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9647 \pgftransformshift
9648 {
9649     \pgfpoint
9650     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9651     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9652 }
9653 \str_if_empty:NTF \l_@@_submatrix_name_str
9654 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9655 {
9656     \@@_node_right:nnnn #2
9657     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9658 }

```

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`.

```

9659 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9660 \flag_clear_new:N \l_@@_code_flag
9661 \l_@@_code_tl
9662 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9663 \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.

```

9664 \cs_new:Npn \@@_pgfpointanchor:n #1
9665 { \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`).

```

9666 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9667 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9668 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9669 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9670 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9671 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9672 { \@@_pgfpointanchor_ii:n { #1 } }
9673 }

```

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`).

```

9674 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9675 { \@@_pgfpointanchor_ii:n { #1 } }

```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nNTF` of the package `etl` but, as of now, we do not load `etl`.

```

9676 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9677 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9678 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9679 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9680 { \@@_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`).

```

9681 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9682 }

```

The following function is for the case when the name contains an hyphen.

```

9683 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9684 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9685 \@@_env:
9686 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9687 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9688 }

```

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.

```

9689 \tl_const:Nn \c_@@_integers_alist_tl
9690 {
9691 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9692 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9693 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9694 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9695 }

```

```

9696 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9697 {

```

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`.

```

9698 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9699 {
9700 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9701 \@@_env: -
9702 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9703 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9704 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9705 }
9706 {
9707 \str_if_eq:eeTF { #1 } { last }
9708 {
9709 \flag_raise:N \l_@@_code_flag
9710 \@@_env: -
9711 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9712 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9713 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9714 }
9715 { #1 }
9716 }
9717 }

```

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`).

```

9718 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9719 {
9720 \pgfnode
9721 { rectangle }
9722 { east }
9723 {
9724 \nullfont
9725 $ % $
9726 \@@_color:o \l_@@_delimiters_color_tl
9727 \left #1
9728 \vcenter
9729 {
9730 \nullfont
9731 \hrule \@height \l_tmpa_dim
9732 \@depth \c_zero_dim
9733 \@width \c_zero_dim
9734 }
9735 \right .
9736 $ % $
9737 }
9738 { #2 }
9739 { }
9740 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

9741 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9742 {
9743   \pgfnode
9744     { rectangle }
9745     { west }
9746     {
9747       \nullfont
9748       $ % $
9749       \colorlet { current-color } { . }
9750       \@@_color:o \l_@@_delimiters_color_tl
9751       \left .
9752       \vcenter
9753         {
9754           \nullfont
9755           \hrule \@height \l_tmpa_dim
9756             \@depth \c_zero_dim
9757             \@width \c_zero_dim
9758         }
9759       \right #1
9760       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9761       ^ { \color { current-color } \smash { #4 } }
9762       $ % $
9763     }
9764     { #2 }
9765     { }
9766 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9767 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9768 {
9769   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9770   \ignorespaces
9771 }
9772 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9773 {
9774   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9775   \ignorespaces
9776 }
9777 \keys_define:nn { nicematrix / Brace }
9778 {
9779   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9780   left-shorten .value_forbidden:n = true ,
9781   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9782   right-shorten .value_forbidden:n = true ,
9783   shorten .meta:n = { left-shorten , right-shorten } ,
9784   shorten .value_forbidden:n = true ,
9785   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9786   color .tl_set:N = \l_tmpa_tl ,
9787   color .value_required:n = true ,
9788   unknown .code:n =
9789     \@@_unknown_key:nn
9790     { nicematrix / Brace }

```

```

9791 { Unknown-key-for-Brace }
9792 }

```

#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`.

```

9793 \cs_new_protected:Npn \l_@@_brace:nnnnn #1 #2 #3 #4 #5
9794 {
9795   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9796   \l_@@_compute_i_j:nn { #1 } { #2 }
9797   \bool_lazy_or:nnTF
9798     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9799     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9800   {
9801     \str_if_eq:eeTF { #5 } { under }
9802     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9803     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9804   }
9805   {
9806     \tl_clear:N \l_tmpa_tl
9807     \keys_set:nn { nicematrix / Brace } { #4 }
9808     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9809     \pgfpicture
9810     \pgfrememberpicturepositiononpagetrue
9811     \pgf@relevantforpicturesizefalse
9812     \bool_if:NT \l_@@_brace_left_shorten_bool
9813     {
9814       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9815       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9816       {
9817         \cs_if_exist:cT
9818           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9819         {
9820           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9821
9822           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9823             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9824         }
9825       }
9826     }
9827     \bool_lazy_or:nnT
9828       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9829       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9830     {
9831       \@@_qpoint:n { col - \l_@@_first_j_tl }
9832       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9833     }
9834     \bool_if:NT \l_@@_brace_right_shorten_bool
9835     {
9836       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9837       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9838       {
9839         \cs_if_exist:cT
9840           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9841         {
9842           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9843           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9844             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9845         }
9846       }
9847     }
9848     \bool_lazy_or:nnT

```

```

9849         { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9850         { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9851         {
9852             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9853             \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9854         }
9855         \pgfset { inner~sep = \c_zero_dim }
9856         \str_if_eq:eeTF { #5 } { under }
9857             { \@@_underbrace_i:n { #3 } }
9858             { \@@_overbrace_i:n { #3 } }
9859         \endpgfpicture
9860     }
9861 \group_end:
9862 }

```

The argument is the text to put above the brace.

```

9863 \cs_new_protected:Npn \@@_overbrace_i:n #1
9864 {
9865     \@@_qpoint:n { row - \l_@@_first_i_tl }
9866     \pgftransformshift
9867     {
9868         \pgfpoint
9869         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9870         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9871     }
9872     \pgfnode
9873     { rectangle }
9874     { south }
9875     {
9876         \vtop
9877         {
9878             \group_begin:
9879             \everycr { }
9880             \halign
9881             {
9882                 \hfil ## \hfil \crcr
9883                 \bool_if:NTF \l_@@_tabular_bool
9884                     { \begin { tabular } { c } #1 \end { tabular } }
9885                     { $ \begin { array } { c } #1 \end { array } $ }
9886                 \cr
9887                 $ % $
9888                 \overbrace
9889                 {
9890                     \hbox_to_wd:nn
9891                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9892                     { }
9893                 }
9894                 $ % $
9895                 \cr
9896             }
9897             \group_end:
9898         }
9899     }
9900     { }
9901     { }
9902 }

```

The argument is the text to put under the brace.

```

9903 \cs_new_protected:Npn \@@_underbrace_i:n #1
9904 {
9905     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9906     \pgftransformshift
9907     {

```

```

9908     \pgfpoint
9909     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9910     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9911 }
9912 \pgfnode
9913 { rectangle }
9914 { north }
9915 {
9916     \group_begin:
9917     \everycr { }
9918     \vbox
9919     {
9920         \halign
9921         {
9922             \hfil ## \hfil \crcr
9923             $ % $
9924             \underbrace
9925             {
9926                 \hbox_to_wd:nn
9927                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9928                 { }
9929             }
9930             $ % $
9931             \cr
9932             \bool_if:NTF \l_@@_tabular_bool
9933             { \begin { tabular } { c } #1 \end { tabular } }
9934             { $ \begin { array } { c } #1 \end { array } $ }
9935             \cr
9936         }
9937     }
9938     \group_end:
9939 }
9940 { }
9941 { }
9942 }

```

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`).

```

9943 \AddToHook { package / tikz / after }
9944 {
9945     \tikzset
9946     {
9947         nicematrix / brace / .style =
9948         {
9949             decoration = { brace , raise = -0.15 em } ,
9950             decorate ,
9951         } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9952         nicematrix / mirrored-brace / .style =
9953         {
9954             nicematrix / brace ,
9955             decoration = mirror ,

```

```

9956     }
9957   }
9958 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9959 \keys_define:nn { nicematrix / Hbrace }
9960 {
9961   color .code:n = ,
9962   horizontal-label .code:n = ,
9963   horizontal-labels .code:n = ,
9964   shorten .code:n = ,
9965   shorten-start .code:n = ,
9966   shorten-end .code:n = ,
9967   shorten+ .code:n = ,
9968   shorten-start+ .code:n = ,
9969   shorten-end+ .code:n = ,
9970   shorten~+ .code:n = ,
9971   shorten-start~+ .code:n = ,
9972   shorten-end~+ .code:n = ,
9973   brace-shift .code:n = ,
9974   brace-shift+ .code:n = ,
9975   brace-shift~+ .code:n = ,
9976   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9977 }

```

Here we need an “fully expandable” command.

```

9978 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9979 {
9980   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9981     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9982     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9983 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9984 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9985 {
9986   \int_compare:nNnTF \c@iRow < { 2 }
9987   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9988     \str_if_eq:nnTF { #2 } { * }
9989     {
9990       \bool_set_true:N \l_@@_nullify_dots_bool
9991       \Ldots
9992       [
9993         line-style = nicematrix / brace ,
9994         #1 ,
9995         up =
9996         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9997       ]
9998     }
9999     {
10000       \Hdotsfor
10001       [
10002         line-style = nicematrix / brace ,
10003         #1 ,
10004         up =
10005         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10006       ]
10007       { #2 }

```

```

10008     }
10009   }
10010   {
10011     \str_if_eq:nnTF { #2 } { * }
10012     {
10013       \bool_set_true:N \l_@@_nullify_dots_bool
10014       \Ldots
10015       [
10016         line-style = nicematrix / mirrored-brace ,
10017         #1 ,
10018         down =
10019           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10020       ]
10021     }
10022     {
10023       \Hdotsfor
10024       [
10025         line-style = nicematrix / mirrored-brace ,
10026         #1 ,
10027         down =
10028           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10029       ]
10030       { #2 }
10031     }
10032   }
10033   \keys_set:nn { nicematrix / Hbrace } { #1 }
10034 }

10035 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
10036 {
10037   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10038   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10039   { \@@_error:nn { Hbrace-not-allowed } { \Vbrace } }
10040 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

10041 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
10042 {
10043   \int_compare:nNnTF \c@jCol < { 2 }
10044   {
10045     \str_if_eq:nnTF { #2 } { * }
10046     {
10047       \bool_set_true:N \l_@@_nullify_dots_bool
10048       \Vdots
10049       [
10050         Vbrace ,
10051         line-style = nicematrix / mirrored-brace ,
10052         #1 ,
10053         down =
10054           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10055       ]
10056     }
10057     {
10058       \Vdotsfor
10059       [
10060         Vbrace ,
10061         line-style = nicematrix / mirrored-brace ,
10062         #1 ,
10063         down =
10064           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10065       ]
10066       { #2 }

```

```

10067     }
10068   }
10069   {
10070     \str_if_eq:nnTF { #2 } { * }
10071     {
10072       \bool_set_true:N \l_@@_nullify_dots_bool
10073       \Vdots
10074       [
10075         Vbrace ,
10076         line-style = nicematrix / brace ,
10077         #1 ,
10078         up =
10079         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10080       ]
10081     }
10082     {
10083       \Vdotsfor
10084       [
10085         Vbrace ,
10086         line-style = nicematrix / brace ,
10087         #1 ,
10088         up =
10089         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10090       ]
10091       { #2 }
10092     }
10093   }
10094   \keys_set:nn { nicematrix / Hbrace } { #1 }
10095 }

```

35 The command TikzEveryCell

```

10096 \bool_new:N \l_@@_not_empty_bool
10097 \bool_new:N \l_@@_empty_bool
10098
10099 \keys_define:nn { nicematrix / TikzEveryCell }
10100 {
10101   not-empty .code:n =
10102     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10103     { \bool_set_true:N \l_@@_not_empty_bool }
10104     { \@@_error:n { detection-of-empty-cells } } ,
10105   not-empty .value_forbidden:n = true ,
10106   empty .code:n =
10107     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10108     { \bool_set_true:N \l_@@_empty_bool }
10109     { \@@_error:n { detection-of-empty-cells } } ,
10110   empty .value_forbidden:n = true ,
10111   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10112 }
10113
10114
10115 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10116 {
10117   \IfPackageLoadedTF { tikz }
10118   {
10119     \group_begin:
10120     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

10121     \tl_set:Nn \l_tmpa_tl { { #2 } }
10122     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10123       { \@@_for_a_block:nnnnn #1 }
10124     \@@_all_the_cells:
10125     \group_end:
10126   }
10127   { \@@_error:n { TikzEveryCell~without~tikz } }
10128 }
10129
10130
10131 \cs_new_protected:Nn \@@_all_the_cells:
10132 {
10133   \int_step_inline:nn \c@iRow
10134   {
10135     \int_step_inline:nn \c@jCol
10136     {
10137       \cs_if_exist:cF { cell - ##1 - #####1 }
10138       {
10139         \clist_if_in:Nef \l_@@_corners_cells_clist
10140           { ##1 - #####1 }
10141         {
10142           \bool_set_false:N \l_tmpa_bool
10143           \cs_if_exist:cTF
10144             { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10145             {
10146               \bool_if:NF \l_@@_empty_bool
10147               { \bool_set_true:N \l_tmpa_bool }
10148             }
10149             {
10150               \bool_if:NF \l_@@_not_empty_bool
10151               { \bool_set_true:N \l_tmpa_bool }
10152             }
10153           \bool_if:NT \l_tmpa_bool
10154           {
10155             \@@_block_tikz:nnnnn
10156             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10157           }
10158         }
10159       }
10160     }
10161   }
10162 }
10163
10164 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10165 {
10166   \bool_if:NF \l_@@_empty_bool
10167   {
10168     \@@_block_tikz:nnnnn
10169     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10170   }
10171   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10172 }
10173
10174 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10175 {
10176   \int_step_inline:nnn { #1 } { #3 }
10177   {
10178     \int_step_inline:nnn { #2 } { #4 }
10179     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10180   }
10181 }

```

36 The key draw-trees-in-col

```

10182 \cs_new_protected:Npn \@@_draw_trees:
10183 {
10184   \bool_if:NTF \l_@@_no_cell_nodes_bool
10185     { \@@_error:nn { draw-trees~with~no-cell-nodes } }
10186     \@@_draw_trees_i:
10187 }
10188 \cs_new_protected:Npn \@@_draw_trees_i:
10189 {
10190   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10191   \dim_zero_new:N \l_@@_em_dim
10192   \dim_set:Nn \l_@@_em_dim { 1 em }
10193   \dim_zero_new:N \l_@@_ex_dim
10194   \dim_set:Nn \l_@@_ex_dim { 1 ex }
10195   \pgfpicture
10196   \pgfrememberpicturepositiononpagetrue
10197   \pgf@relevantforpicturesizefalse
10198   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10199     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10200   \@@_color:o \l_@@_trees_color_tl
10201   \pgfsetcornersarced
10202     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10203   \clist_map_function:NN \g_@@_col_with_trees_clist
10204     \@@_draw_trees_in_col:n
10205   \clist_gclear:N \g_@@_col_with_trees_clist
10206   \endpgfpicture
10207 }
10208 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10209 {

```

The argument is provided by curryfication.

```

10210   \int_compare:nNnTF { #1 } > \c@jCol
10211     { \@@_error:nn { Col~outside~tabular~in~trees } }
10212     {
10213       \int_compare:nNnTF { #1 } = \c@jCol
10214         { \@@_error:nn { Last~col~in~trees } }
10215         \@@_draw_trees_in_col_i:n
10216     }
10217   { #1 }
10218 }
10219 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10220 {
10221   \int_set:Nn \l_tmpa_int { 1 }
10222   \int_step_inline:nn { \c@iRow + 1 }
10223   {
10224     \cs_if_exist:cT
10225       { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10226       {
10227         \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10228         {

```

Now, you will, potentially, draw a tree.

```

10229       \@@_draw_tree:nee
10230       { #1 }
10231       { \int_use:N \l_tmpa_int }
10232       { \int_eval:n { ##1 - 1 } }
10233     }
10234   \int_set:Nn \l_tmpa_int { ##1 }
10235 }

```

```

10236     }
10237     \int_compare:nNnT \c@iRow > \l_tmpa_int
10238     {
10239         \@@_draw_tree:nee
10240         { #1 }
10241         { \int_use:N \l_tmpa_int }
10242         { \int_eval:n { \c@iRow } }
10243     }
10244 }

```

#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.

```

10245 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10246 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10247     \pgfpointanchor { \@@_env: - #1 } { 5 }
10248     \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.

```

10249     \bool_set_false:N \l_tmpa_bool
10250     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10251     {
10252         \cs_if_exist:cT
10253         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10254     {
10255         \pgfpointanchor
10256         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10257         { base~west }
10258         \pgfpathmoveto
10259         {
10260             \pgfpoint
10261             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10262             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10263         }
10264         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10265         \bool_if:NF \l_tmpa_bool
10266         {
10267             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10268             \pgfpathlineto
10269             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10270             \bool_set_true:N \l_tmpa_bool
10271         }
10272         \pgfusepath { stroke }
10273     }
10274 }
10275 }
10276 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10277 \cs_new_protected:Npn \@@_create_blocks_in_col:
10278 {
10279     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10280     \clist_map_inline:Nn \g_@@_cbic_clist
10281     {
10282         \cs_set:cpn
10283         {
10284             pgf @ sh @ ns @ \@@_env:
10285             - \int_eval:n { \c@iRow + 1 } - ##1 }

```

```

10286         { rien }
\l_tmpa_int will be the first row of the block being created.
10287     \int_set:Nn \l_tmpa_int { 1 }
10288     \int_step_inline:nn { \c@iRow + 1 }
10289     {
10290         \cs_if_exist:cT
10291         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10292         {
10293             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10294             {
10295                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10296                 {
10297                     { \int_use:N \l_tmpa_int }
10298                     { ##1 }
10299                     { \int_eval:n { #####1 - 1 } }
10300                     { ##1 }
10301                     { }
10302                 }
10303             }
10304             \int_set:Nn \l_tmpa_int { #####1 }
10305         }
10306     }
10307 }
10308 \clist_gclear:N \g_@@_cbic_clist
10309 }

```

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.

```

10310 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10311 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10312 \NewDocumentCommand \@@_ShowCellNames { }
10313 {
10314     \bool_if:NT \l_@@_in_code_after_bool
10315     {
10316         \pgfpicture
10317         \pgfrememberpicturepositiononpagetrue
10318         \pgf@relevantforpicturesizefalse
10319         \pgfpathrectanglecorners
10320         { \@@_qpoint:n { 1 } }
10321         { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10322         \pgfsetfillopacity { 0.75 }
10323         \pgfsetfillcolor { white }
10324         \pgfusepathqfill
10325         \endpgfpicture
10326     }
10327     \dim_gzero_new:N \g_@@_tmpc_dim
10328     \dim_gzero_new:N \g_@@_tmpd_dim
10329     \dim_gzero_new:N \g_@@_tmpe_dim
10330     \int_step_inline:nn \c@iRow
10331     {
10332         \bool_if:NTF \l_@@_in_code_after_bool
10333         {
10334             \pgfpicture
10335             \pgfrememberpicturepositiononpagetrue

```

```

10336         \pgf@relevantforpicturesizefalse
10337     }
10338     { \begin { pgfpicture } }
10339     \@@_qpoint:n { row - ##1 }
10340     \dim_set_eq:NN \l_tmpa_dim \pgf@y
10341     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10342     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10343     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10344     \bool_if:NTF \l_@@_in_code_after_bool
10345     { \endpgfpicture }
10346     { \end { pgfpicture } }
10347     \int_step_inline:nn \c@jCol
10348     {
10349         \hbox_set:Nn \l_tmpa_box
10350         {
10351             \normalfont \Large \sffamily \bfseries
10352             \bool_if:NTF \l_@@_in_code_after_bool
10353             { \color { red } }
10354             { \color { red ! 50 } }
10355             ##1 - ####1
10356         }
10357         \bool_if:NTF \l_@@_in_code_after_bool
10358         {
10359             \pgfpicture
10360             \pgfrememberpicturerepositiononpagetrue
10361             \pgf@relevantforpicturesizefalse
10362         }
10363         { \begin { pgfpicture } }
10364         \@@_qpoint:n { col - ####1 }
10365         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10366         \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10367         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10368         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10369         \bool_if:NTF \l_@@_in_code_after_bool
10370         { \endpgfpicture }
10371         { \end { pgfpicture } }
10372         \fp_set:Nn \l_tmpa_fp
10373         {
10374             \fp_min:nn
10375             {
10376                 \fp_min:nn
10377                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10378                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10379             }
10380             { 1.0 }
10381         }
10382         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10383         \pgfpicture
10384         \pgfrememberpicturerepositiononpagetrue
10385         \pgf@relevantforpicturesizefalse
10386         \pgftransformshift
10387         {
10388             \pgfpoint
10389             { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10390             \g_tmpa_dim
10391         }
10392         \pgfnode
10393         { rectangle }
10394         { center }
10395         { \box_use:N \l_tmpa_box }
10396         { }
10397         { }
10398         \endpgfpicture

```

```

10399     }
10400   }
10401 }

```

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.

```

10402 \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.

```

10403 \bool_new:N \g_@@_footnote_bool

10404 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10405 {
10406   You-have-used-the-key~' \l_keys_key_str '~when~loading~nicematrix~
10407   but~that~key~is~unknown. \\
10408   It~will~be~ignored. \\
10409   For~a~list~of~the~available~keys,~type~H~<return>.
10410 }
10411 {
10412   The~available~keys~are~(in~alphabetic~order):~
10413   footnote,~
10414   footnotehyper,~
10415   messages-for-Overleaf,~
10416   renew-dots~and~
10417   renew-matrix.
10418 }

10419 \keys_define:nn { nicematrix }
10420 {
10421   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10422   renew-dots .value_forbidden:n = true ,
10423   renew-matrix .code:n = \@@_renew_matrix: ,
10424   renew-matrix .value_forbidden:n = true ,
10425   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10426   footnote .bool_set:N = \g_@@_footnote_bool ,
10427   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10428   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10429 }
10430 \ProcessKeyOptions

10431 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10432 {
10433   You~can't~use~the~option~'footnote'~because~the~package~
10434   footnotehyper~has~already~been~loaded.~
10435   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10436   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10437   of~the~package~footnotehyper.\\
10438   The~package~footnote~won't~be~loaded.
10439 }

```

```

10440 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10441 {
10442   You~can't~use~the~option~'footnotehyper'~because~the~package~
10443   footnote~has~already~been~loaded.~
10444   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10445   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10446   of~the~package~footnote.\\
10447   The~package~footnotehyper~won't~be~loaded.
10448 }

```

```

10449 \bool_if:NT \g_@@_footnote_bool
10450 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10451   \IfClassLoadedTF { beamer }
10452   { \bool_set_false:N \g_@@_footnote_bool }
10453   {
10454     \IfPackageLoadedTF { footnotehyper }
10455     { \@@_error:n { footnote~with~footnotehyper~package } }
10456     { \usepackage { footnote } }
10457   }
10458 }

```

```

10459 \bool_if:NT \g_@@_footnotehyper_bool
10460 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10461   \IfClassLoadedTF { beamer }
10462   { \bool_set_false:N \g_@@_footnote_bool }
10463   {
10464     \IfPackageLoadedTF { footnote }
10465     { \@@_error:n { footnotehyper~with~footnote~package } }
10466     { \usepackage { footnotehyper } }
10467   }
10468   \bool_set_true:N \g_@@_footnote_bool
10469 }

```

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.

```

10470 \bool_new:N \l_@@_underscore_loaded_bool
10471 \IfPackageLoadedT { underscore }
10472 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10473 \AtBeginDocument
10474 {
10475   \bool_if:NF \l_@@_underscore_loaded_bool
10476   {
10477     \IfPackageLoadedT { underscore }
10478     { \@@_error:n { underscore~after~nicematrix } }
10479   }
10480 }

```

41 Compatibility with threeparttable

```
10481 \AtBeginDocument
10482 {
10483   \IfPackageLoadedT { threeparttable }
10484   {
10485     \AddToHook { env / threeparttable / begin }
10486     {
10487       \TPT@hookin { NiceTabular }
10488       \TPT@hookin { NiceTabular* }
10489       \TPT@hookin { NiceTabularX }
10490     }
10491   }
10492 }
```

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.

```
10493 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10494 {
10495   \str_if_eq:eeTF
10496   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10497   { + }
10498   {
10499     \str_set:Ne \l_tmpa_str
10500     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10501     \bool_set_false:N \l_tmpa_bool
10502     \clist_map_inline:nn { #1 }
10503     {
10504       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10505       {
10506         \@@_error:n { key~without~+~exists }
10507         \bool_set_true:N \l_tmpa_bool
10508         \clist_map_break:
10509       }
10510     }
10511     \bool_if:NF \l_tmpa_bool
10512     {
10513       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10514       \@@_unknown_key_i:nn { #1 } { #2 }
10515     }
10516   }
10517   { \@@_unknown_key_i:nn { #1 } { #2 } }
10518 }
```

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.

```
10519 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10520 {
10521   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10522   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10523   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10524   \bool_set_false:N \l_tmpa_bool
```

```

10525 \clist_map_inline:nn { #1 }
10526 {
10527     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10528     {
10529         \@@_error:n { key~with~normal~form~exists }
10530         \bool_set_true:N \l_tmpa_bool
10531         \clist_map_break:
10532     }
10533 }
10534 \bool_if:NF \l_tmpa_bool
10535 {
10536     \@@_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).

```

10537     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10538     { \msg_info:nn { nicematrix } { #2~+ } }
10539 }
10540 }
10541 \@@_msg_new:nn { key~without~+~exists }
10542 {
10543     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10544     additive~syntax~(with~+=).~It~will~be~ignored.
10545 }
10546 \@@_msg_new:nn { key~with~normal~form~exists }
10547 {
10548     No~key~'\l_keys_key_str'.\\
10549     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10550 }
10551 \str_const:Ne \c_@@_available_keys_str
10552 {
10553     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10554     { For~a~list~of~the~available~keys,~type~H~<return>. }
10555 }
10556 \seq_new:N \g_@@_types_of_matrix_seq
10557 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10558 {
10559     NiceMatrix ,
10560     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10561 }
10562 \seq_gset_map_e:Nn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10563 { \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`.

```

10564 \cs_new_protected:Npn \@@_err_too_many_cols:
10565 {
10566     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10567     { \@@_fatal:nn { too-many-cols-for-array } }
10568     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10569     { \@@_fatal:n { too-many-cols-for-matrix } }
10570     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10571     { \@@_fatal:n { too-many-cols-for-matrix } }
10572     \bool_if:NF \l_@@_last_col_without_value_bool
10573     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10574 }

```

The following command must *not* be protected since it's used in an error message.

```

10575 \cs_new:Npn \@@_message_hdotsfor:
10576 {
10577   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10578   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10579     \token_to_str:N \Hbrace \ is~incorrect. }
10580 }

10581 \cs_new_protected:Npn \@@_Hline_in_cell:
10582 { \@@_error:n { Misuse~of~Hline } }

10583 \@@_msg_new:nn { Misuse~of~Hline }
10584 {
10585   Misuse~of~\token_to_str:N \Hline. \\
10586   Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:.~
10587   \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must~be~used~only~
10588   at~the~beginning~of~a~row.~Your~command~will~be~ignored.
10589 }

10590 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
10591 {
10592   Incompatible~options.\\
10593   You~should~not~use~'hvlines',~'rounded~corners'~and~'corners'~at~the~same~time.~
10594   The~output~will~not~be~reliable.
10595 }

10596 \@@_msg_new:nn { Body~alone }
10597 {
10598   \token_to_str:N \Body\ alone. \\
10599   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10600 }

10601 \@@_msg_new:nn { Bad~use~of~CodeBefore }
10602 {
10603   Bad~use~of~\token_to_str:N \CodeBefore. \\
10604   \token_to_str:N \CodeBefore\ must~be~used~only~at~the~beginning~of~
10605   the~environment.
10606 }

10607 \@@_msg_new:nn { NiceTabularX~probably~required }
10608 {
10609   Incorrect~syntax.\\
10610   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10611   ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10612   with~the~key~'width'.
10613 }

10614 \@@_msg_new:nn { cellcolor~in~Block }
10615 {
10616   Bad~use~of~\token_to_str:N \cellcolor \\
10617   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10618   \bool_if:NTF \l_@@_amp_in_blocks_bool
10619     { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10620     { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10621   .~Here,~you~should~use~the~key~'fill'~of~the~block.~Your~command~will~be~ignored.
10622 }

10623 \@@_msg_new:nn { rowcolor~in~Block }
10624 {
10625   Bad~use~of~\token_to_str:N \rowcolor \\
10626   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10627   However,~it's~possible~to~color~the~block~with~its~key~'fill'.~
10628   Your~command~will~be~ignored.
10629 }

10630 \@@_msg_new:nn { key~color~inside }
10631 {
10632   Deleted~key.\\

```

```

10633     The-key~'color-inside'~(and-its-alias~'colortbl-like')~has-been-deleted-in
10634     ~'nicematrix'~and~must~not~be~used.
10635 }

10636 \@@_msg_new:nn { Invalid~argument~for~w }
10637 {
10638     Invalid~argument~for~w.\\
10639     You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10640     but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10641     If~you~go~on,~'c'~will~be~used.
10642 }

10643 \@@_msg_new:nn { invalid~weight }
10644 {
10645     Unknown~key.\\
10646     The-key~'\l_keys_key_str'~of~your~column~X~is~unknown~and~will~be~ignored.~
10647     The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10648     \IfPackageLoadedTF { varwidth }
10649     { (since~'varwidth'~is~loaded)~}
10650     { (if~you~load~'varwidth')~}
10651     and~real~numbers~for~the~weight~of~the~X~column.
10652 }

10653 \@@_msg_new:nn { last~col~not~used }
10654 {
10655     Column~not~used.\\
10656     The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
10657     in~your~\@@_full_name_env: .~However,~you~can~go~on.
10658 }

10659 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10660 {
10661     Too~many~columns.\\
10662     In~the~row~ \int_eval:n { \c@iRow },~
10663     you~try~to~use~more~columns~
10664     than~allowed~by~your~ \@@_full_name_env: .
10665     \@@_message_hdotsfor: \
10666     The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10667     (plus~the~exterior~columns).~
10668     But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10669 }

10670 \@@_msg_new:nn { too~many~cols~for~matrix }
10671 {
10672     Too~many~columns.\\
10673     In~the~row~ \int_eval:n { \c@iRow },~
10674     you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10675     \@@_message_hdotsfor: \
10676     Recall~that~the~maximal~number~of~columns~for~a~matrix~
10677     (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10678     LaTeX~counter~'MaxMatrixCols'.~
10679     Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10680     (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10681     But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10682 }

10683 \@@_msg_new:nn { too~many~cols~for~array }
10684 {
10685     Too~many~columns.\\
10686     In~the~row~ \int_eval:n { \c@iRow },~
10687     ~you~try~to~use~more~columns~than~allowed~by~your~
10688     \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10689     \int_use:N \g_@@_static_num_of_col_int \
10690     \bool_if:nT
10691     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10692     { (plus~the~exterior~ones)~}

```

```

10693     since~the~preamble~is~' \g_@@_user_preamble_tl ' .~
10694     But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10695 }

10696 \@@_msg_new:nn { columns~not~used }
10697 {
10698     Columns~not~used.\\
10699     The~preamble~of~your~ \@@_full_name_env: \ is~
10700     '\exp_not:V \g_@@_user_preamble_tl'.~
10701     It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10702     columns~but~you~only~used~ \int_use:N \c@jCol .~The~columns~you~did~not~
10703     used~won't~be~created.~You~won't~have~similar~warning~till~the~end~of~the~document.
10704 }

10705 }

10706 \@@_msg_new:nn { Bad~use~of~NiceTabularNotes }
10707 {
10708     Bad~use~of~\token_to_str:N \NiceTabularNotes \\
10709     \token_to_str:N~\NiceTabularNotes\ should~be~used~only~after~at~tabular~
10710     which~uses~`notes/no-print`.~ Your~command~will~be~ignored.
10711 }

10712 \@@_msg_new:nn { empty~preamble }
10713 {
10714     Empty~preamble.\\
10715     The~preamble~of~your~ \@@_full_name_env: \ is~empty.
10716 }

10717 \@@_msg_new:nn { in~first~col }
10718 {
10719     Erroneous~use.\\
10720     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.~
10721     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-col'.
10722 }

10723 \@@_msg_new:nn { in~last~col }
10724 {
10725     Erroneous~use.\\
10726     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.~
10727     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-col'.
10728 }

10729 \@@_msg_new:nn { in~first~row }
10730 {
10731     Erroneous~use.\\
10732     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
10733     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-row'.
10734 }

10735 \@@_msg_new:nn { in~last~row }
10736 {
10737     Erroneous~use.\\
10738     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
10739     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-row'.
10740 }

10741 \@@_msg_new:nn { TopRule~without~booktabs }
10742 {
10743     Erroneous~use.\\
10744     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
10745     You~should~load~'booktabs'~(before~or~after~'nicematrix').~
10746     Your~command~will~be~ignored.
10747 }

10748 \@@_msg_new:nn{ rotate~in~p~col }
10749 {
10750     \token_to_str:N \rotate\ forbidden.\\
10751     You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10752     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~

```

```

10753     If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
10754     consider~the~classical~command~\token_to_str:N \rotatebox.
10755 }
10756 \@@_msg_new:nn { TopRule~without~tikz }
10757 {
10758     Erroneous~use.\\
10759     You~can't~use~the~command~#1~because~TikZ~is~not~loaded.~
10760     You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
10761     \IfPackageLoadedF { booktabs }
10762         { You~should~also~load~'booktabs'~
10763           with~\token_to_str:N \usepackage \{booktabs\}.~ }
10764     Your~command~will~be~ignored.
10765 }
10766 \@@_msg_new:nn { caption~outside~float }
10767 {
10768     Key~caption~forbidden.\\
10769     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10770     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
10771 }
10772 \@@_msg_new:nn { short~caption~without~caption }
10773 {
10774     You~should~not~use~the~key~'short~caption'~without~'caption'.~
10775     However,~your~'short~caption'~will~be~used~as~'caption'.
10776 }
10777 \@@_msg_new:nn { double~closing~delimiter }
10778 {
10779     Double~delimiter.\\
10780     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10781     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
10782     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10783 }
10784 \@@_msg_new:nn { delimiter~after~opening }
10785 {
10786     Double~delimiter.\\
10787     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10788     delimiter.~That~delimiter~will~be~ignored.
10789 }
10790 \@@_msg_new:nn { bad~option~for~line~style }
10791 {
10792     Bad~line~style.\\
10793     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10794     is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
10795     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
10796 }
10797 \@@_msg_new:nn { draw~trees~with~no~cell~nodes }
10798 {
10799     Incompatible~keys.\\
10800     You~can't~use~the~key~'draw~trees~in~col'~here~because~the~key~'no~cell~nodes'~
10801     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10802     is~to~speed~compilation~up).~If~you~go~on,~that~key~will~be~ignored.
10803 }
10804 \@@_msg_new:nn { corners~with~no~cell~nodes }
10805 {
10806     Incompatible~keys.\\
10807     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10808     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10809     is~to~speed~compilation~up).\\
10810     If~you~go~on,~that~key~will~be~ignored.
10811 }
10812 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }

```

```

10813 {
10814     Incompatible-keys.\
10815     You-can't~create~'extra-nodes'~here~because~the~key~'no-cell-nodes'~
10816     is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10817     is~to~speed~compilation~up).~If~you~go~on,~those~extra-nodes~won't~be~created.
10818 }
10819 \@@_msg_new:nn { Identical-notes-in-caption }
10820 {
10821     Identical~tabular-notes.\
10822     You-can't~put~several~notes~with~the~same~content~in~
10823     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
10824     If~you~go~on,~the~output~will~probably~be~erroneous.
10825 }
10826 \@@_msg_new:nn { tabularnote~below~the~tabular }
10827 {
10828     \token_to_str:N \tabularnote \ forbidden\
10829     You-can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10830     of~your~tabular~because~the~caption~will~be~composed~below~
10831     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10832     key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .~
10833     Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
10834     no~similar~error~will~raised~in~this~document.
10835 }
10836 \@@_msg_new:nn { Unknown-key~for~rules }
10837 {
10838     Unknown~key.\
10839     There~are~only~three~keys~available~here:~'width',~'color'~and~
10840     'fix-vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10841 }
10842 \@@_msg_new:nn { Unknown-key~for~trees }
10843 {
10844     Unknown~key.\
10845     There~are~only~three~keys~available~here:~width~color~and~
10846     rounded-corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
10847 }
10848 % \end{macrocode}
10849 %
10850 %
10851 % \begin{macrocode}
10852 \@@_msg_new:nn { Unknown-key~for~Hbrace }
10853 {
10854     Unknown~key.\
10855     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10856     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10857     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10858     'horizontal-label(s)',~'shorten'~'shorten-end'~
10859     and~'shorten-start'.
10860 }
10861 \@@_msg_new:nn { Unknown-key~for~TikzEveryCell }
10862 {
10863     Unknown~key.\
10864     There~is~only~two~keys~available~here:~
10865     'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10866 }
10867 \@@_msg_new:nn { Unknown-key~for~rotate }
10868 {
10869     Unknown~key.\
10870     The~only~keys~available~here~are~'c'~and~'-90'.~
10871     Your~key~' \l_keys_key_str '~will~be~ignored.
10872 }
10873 \@@_msg_new:nnn { Unknown-key~for~custom-line }

```

```

10874 {
10875   Unknown~key.\\
10876   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10877   It~you~go~on,~you~will~probably~have~other~errors. \\
10878   \c_@@_available_keys_str
10879 }
10880 {
10881   The~available~keys~are~(in~alphabetic~order):~
10882   ccommand,~
10883   color,~
10884   command,~
10885   dotted,~
10886   letter,~
10887   multiplicity,~
10888   sep~color,~
10889   tikz,~and~total~width.
10890 }
10891 \@@_msg_new:nnn { Unknown~key~for~default~line }
10892 {
10893   Unknown~key.\\
10894   The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10895   It~you~go~on,~you~will~probably~have~other~errors. \\
10896   \c_@@_available_keys_str
10897 }
10898 {
10899   The~available~keys~are~(in~alphabetic~order):~
10900   color,~
10901   dotted,~
10902   multiplicity,~
10903   sep~color,~
10904   tikz~(when~TikZ~is~loaded)~
10905   and~total~width.
10906 }
10907 \@@_msg_new:nnn { Unknown~key~for~xdots }
10908 {
10909   Unknown~key.\\
10910   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10911   \c_@@_available_keys_str
10912 }
10913 {
10914   The~available~keys~are~(in~alphabetic~order):~
10915   'color',~
10916   'horizontal(s)-labels',~
10917   'inter',~
10918   'line~style',~
10919   'nullify',~
10920   'radius',~
10921   'shorten',~
10922   'shorten~end'~and~'shorten~start'.
10923 }
10924 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10925 {
10926   Unknown~key.\\
10927   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
10928   (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
10929 }
10930 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10931 {
10932   Error~with~'draw~trees~in~col' \\
10933   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10934   is~\int~use:N \c@jCol.~It~will~be~ignored.
10935 }

```

```

10936 \@@_msg_new:nn { Last-col-in-trees }
10937 {
10938   Error~with~'draw-trees-in-col' \\
10939   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
10940   the~last~column~of~your~tabular.~It~will~be~ignored.
10941 }
10942 \@@_msg_new:nn { label~without~caption }
10943 {
10944   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10945   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10946 }
10947 \@@_msg_new:nn { W~warning }
10948 {
10949   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10950   (row~ \int_use:N \c@iRow ).
10951 }
10952 \@@_msg_new:nn { Construct~too~large }
10953 {
10954   Construct~too~large.\\
10955   Your~command~ \token_to_str:N #1
10956   can't~be~drawn~because~your~matrix~is~too~small.~
10957   Your~command~will~be~ignored.
10958 }
10959 \@@_msg_new:nn { underscore~after~nicematrix }
10960 {
10961   Problem~with~'underscore'.\\
10962   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10963   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10964   ' \token_to_str:N \Cdots \token_to_str:N _
10965   \{ n \token_to_str:N \text \{ ~times \} \}' .
10966 }
10967 \@@_msg_new:nn { ampersand~in~light-syntax }
10968 {
10969   Ampersand~forbidden.\\
10970   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10971   ~the~key~'light-syntax'~is~in~force.
10972 }
10973 \@@_msg_new:nn { double-backslash~in~light-syntax }
10974 {
10975   Double~backslash~forbidden.\\
10976   You~can't~use~ \token_to_str:N \\
10977   ~to~separate~rows~because~the~key~'light-syntax'~
10978   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10979   (set~by~the~key~'end-of-row').
10980 }
10981 \@@_msg_new:nn { bad~value~for~baseline }
10982 {
10983   Bad~value~for~baseline.\\
10984   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10985   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10986   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10987   the~form~'line-i'.~A~value~of~1~will~be~used.
10988 }
10989 \@@_msg_new:nn { bad~value~for~baseline~line }
10990 {
10991   Bad~value~for~baseline~with~line.\\
10992   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10993   valid.~The~number~of~the~line~must~be~between~1~and~
10994   \int_eval:n { \c@iRow + 1 }.~
10995   A~value~of~'line-1'~will~be~used.
10996 }

```

```

10997 \@@_msg_new:nn { detection-of-empty-cells }
10998 {
10999   Problem-with~'not-empty'\
11000   For-technical-reasons,~you-must-activate~
11001   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
11002   in-order-to-use-the-key~' \l_keys_key_str '~
11003   Your-key-will-be-ignored.
11004 }
11005 \@@_msg_new:nn { siunitx-too-old }
11006 {
11007   siunitx-too-old\
11008   You-can't-use-the-columns~'S'~because-your-version-of~'siunitx'~
11009   is-too-old.~You-need-at-least-the-version~3.5.1~(2026/03/26).
11010 }
11011 \@@_msg_new:nn { Invalid-name }
11012 {
11013   Invalid-name.\
11014   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
11015   \SubMatrix \ of~your~ \@@_full_name_env: .~
11016   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
11017   Your-key-will-be-ignored.
11018 }
11019 \@@_msg_new:nn { Hbrace-not-allowed }
11020 {
11021   Command-not-allowed.\
11022   You-can't-use-the-command~ \token_to_str:N #1
11023   because-you-have-not-loaded~
11024   \IfPackageLoadedTF { tikz }
11025     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
11026     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
11027   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
11028   Your-command-will-be-ignored.
11029 }
11030 \@@_msg_new:nn { Vbrace-not-allowed }
11031 {
11032   Command-not-allowed.\
11033   You-can't-use-the-command~ \token_to_str:N \Vbrace \
11034   because-you-have-not-loaded-TikZ~
11035   and-the-TikZ-library~'decorations.pathreplacing'~
11036   Use: ~\token_to_str:N \usepackage \{tikz\}~
11037   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
11038   Your-command-will-be-ignored.
11039 }
11040 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
11041 {
11042   Wrong-line.\
11043   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
11044   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11045   number~is~not~valid.~It~will~be~ignored.
11046 }
11047 \@@_msg_new:nn { Impossible-SubMatrix }
11048 {
11049   Impossible-SubMatrix.\
11050   It's-impossible-to-draw~your~\token_to_str:N \SubMatrix \
11051   because~all~the~cells~are~empty~in~the~column~on~the~#1~
11052   side~of~that~\token_to_str:N \SubMatrix.
11053   \bool_if:NT \l_@@_submatrix_slim_bool
11054     { ~Maybe-you-should-try-without-the-key~'slim'. } \
11055   This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11056 }
11057 \@@_msg_new:nn { Impossible-SubMatrix-no-cell-nodes }

```

```

11058 {
11059     Impossible~SubMatrix.\\
11060     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
11061     because~the~key~'no-cell-nodes'~is~in~force.~
11062     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11063 }
11064 \@@_msg_new:nnn { width~without~X~columns }
11065 {
11066     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11067     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11068     Your~key~will~be~ignored.
11069 }
11070 {
11071     This~message~is~the~message~'width~without~X~columns'~
11072     of~the~module~'nicematrix'.~
11073     The~experimented~users~can~disable~that~message~with~
11074     \token_to_str:N \msg_redirect_name:nnn .\\
11075 }
11076
11077 \@@_msg_new:nn { key~multiplicity~with~dotted }
11078 {
11079     Incompatible~keys. \\
11080     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11081     in~a~'custom-line'.~They~are~incompatible.~
11082     The~key~'multiplicity'~will~be~ignored.
11083 }
11084 \@@_msg_new:nn { empty~environment }
11085 {
11086     Empty~environment.\\
11087     Your~ \@@_full_name_env: \ is~empty.
11088 }
11089 \@@_msg_new:nn { No~letter~and~no~command }
11090 {
11091     Erroneous~use.\\
11092     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
11093     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11094     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11095 }
11096 \@@_msg_new:nn { Forbidden~letter }
11097 {
11098     Forbidden~letter.\\
11099     You~can't~use~the~letter~'#1'~for~a~customized~line.~
11100     It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11101 }
11102 \@@_msg_new:nn { Several~letters }
11103 {
11104     Wrong~name.\\
11105     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11106     have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11107 }
11108 \@@_msg_new:nn { Delimiter~with~small }
11109 {
11110     Delimiter~forbidden.\\
11111     You~can't~put~a~delimiter~in~the~preamble~of~your~
11112     \@@_full_name_env: \
11113     because~the~key~'small'~is~in~force.
11114 }
11115 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11116 {
11117     Unknown~cell.\\

```

```

11118 Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11119 the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11120 can't~be~executed~because~a~cell~doesn't~exist.~
11121 Your~command~ \token_to_str:N \line \ will~be~ignored.
11122 }
11123 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11124 {
11125 Duplicate~name. \\
11126 The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11127 in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11128 \bool_if:NF \g_@@_messages_for_Overleaf_bool
11129 { For~a~list~of~the~names~already~used,~type~H~<return>. }
11130 }
11131 {
11132 The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11133 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11134 }
11135 \@@_msg_new:nn { r~or~l~with~preamble }
11136 {
11137 Erroneous~use. \\
11138 You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11139 You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11140 your~ \@@_full_name_env: .~
11141 Your~key~will~be~ignored.
11142 }
11143 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11144 {
11145 Erroneous~use. \\
11146 You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11147 in~an~exterior~column~of~the~array.
11148 }
11149 \@@_msg_new:nn { bad~corner }
11150 {
11151 Bad~corner. \\
11152 '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11153 'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11154 This~specification~of~corner~will~be~ignored.
11155 }
11156 \@@_msg_new:nn { bad~border }
11157 {
11158 Bad~border.\\
11159 '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11160 (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11161 The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11162 also~use~the~key~'tikz'
11163 \IfPackageLoadedF { tikz }
11164 { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11165 This~specification~of~border~will~be~ignored.
11166 }
11167 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11168 {
11169 TikZ~not~loaded. \\
11170 You~can't~use~ \token_to_str:N \TikzEveryCell \
11171 because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11172 \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11173 Your~command~will~be~ignored.
11174 }
11175 \@@_msg_new:nn { tikz~key~without~tikz }
11176 {
11177 TikZ~not~loaded. \\
11178 You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N

```

```

11179 \Block '~because-you-have-not-loaded-TikZ.~
11180 You-can-load-TikZ-with-\token_to_str:N \usepackage \{tikz\},~
11181 before-or-after-'nicematrix'.~Your-key-will-be-ignored.
11182 }
11183 \@@_msg_new:nn { Bad~argument~for~Block }
11184 {
11185   Bad~argument. \\
11186   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11187   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11188   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
11189   the~argument~was~empty).
11190 }
11191 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11192 {
11193   Erroneous~use. \\
11194   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11195   'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11196   (the~value~' \l_keys_value_tl '~will~be~ignored).
11197 }
11198 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11199 {
11200   Erroneous~use. \\
11201   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11202   'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11203   (the~value~' \l_keys_value_tl '~will~be~ignored).
11204 }
11205 \@@_msg_new:nn { Block~too~large~1 }
11206 {
11207   Block~too~large. \\
11208   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
11209   too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11210 }
11211 \@@_msg_new:nn { Block~too~large~2 }
11212 {
11213   Block~too~large. \\
11214   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
11215   \g_@@_static_num_of_col_int \
11216   columns~but~you~use~only~ \int_use:N \c_jCol \ and~that's~why~a~block~
11217   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
11218   (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: .~
11219   This~block~and~maybe~others~will~be~ignored.
11220 }
11221 \@@_msg_new:nn { unknown~column~type }
11222 {
11223   Bad~column~type. \\
11224   The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.
11225 }
11226 \@@_msg_new:nn { unknown~column~type~multicolumn }
11227 {
11228   Bad~column~type. \\
11229   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
11230   ~of~your~ \@@_full_name_env: \ is~unknown.
11231 }
11232 \@@_msg_new:nn { unknown~column~type~S }
11233 {
11234   Bad~column~type. \\
11235   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown.~
11236   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11237   load~that~package~with~\token_to_str:N \usepackage \{siunitx\}.
11238 }

```

```

11239 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
11240 {
11241   Bad~column~type. \\
11242   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11243   of~your~ \@@_full_name_env: \ is~unknown.~If~you~want~to~use~the~column~
11244   type~'S'~of~siunitx,~you~should~load~that~package~with~
11245   \token_to_str:N \usepackage \{siunitx\}.
11246 }
11247 \@@_msg_new:nn { tabularnote~forbidden }
11248 {
11249   Forbidden~command. \\
11250   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11251   ~here.~This~command~is~available~only~in~
11252   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11253   the~argument~of~a~command~\token_to_str:N \caption \ included~
11254   in~an~environment~\{table\}.~Your~command~will~be~ignored.
11255 }
11256 \@@_msg_new:nn { borders~forbidden }
11257 {
11258   Forbidden~key.\\
11259   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11260   because~the~option~'rounded~corners'~is~in~force~with~a~non~zero~value.~
11261   Your~key~will~be~ignored.
11262 }
11263 \@@_msg_new:nn { bottomrule~without~booktabs }
11264 {
11265   booktabs~not~loaded.\\
11266   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11267   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11268   after~'nicematrix'.~Your~key~will~be~ignored.
11269 }
11270 \@@_msg_new:nn { enumitem~not~loaded }
11271 {
11272   enumitem~not~loaded. \\
11273   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11274   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11275   (before~or~after~'nicematrix').~All~the~commands~
11276   \token_to_str:N \tabularnote \ will~be~ignored~in~the~document.
11277 }
11278 \@@_msg_new:nn { tikz~without~tikz }
11279 {
11280   TikZ~not~loaded. \\
11281   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11282   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11283   If~you~go~on,~your~key~will~be~ignored.
11284 }
11285 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11286 {
11287   TikZ~not~loaded. \\
11288   You~have~used~the~key~'tikz'~in~the~definition~of~a~
11289   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
11290   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11291   use~that~custom~line.~You~should~load~TikZ~with~
11292   \token_to_str:N \usepackage \{tikz\}.
11293 }
11294 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11295 {
11296   TikZ~not~loaded. \\
11297   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11298   command~ \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11299   Your~key~will~be~ignored.~You~should~load~TikZ~with~

```

```

11300 \token_to_str:N \usepackage \{tikz\}
11301 }
11302 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11303 {
11304   Erroneous~use.\\
11305   In~a~'custom~line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11306   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11307   The~key~'color'~will~be~ignored.
11308 }
11309 \@@_msg_new:nn { Wrong~last~row }
11310 {
11311   Wrong~number.\\
11312   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
11313   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11314   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11315   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11316   problem~by~using~'last~row'~without~value~(more~compilations~
11317   might~be~necessary).
11318 }
11319 \@@_msg_new:nn { Yet~in~env }
11320 {
11321   Nested~environments.\\
11322   Environments~of~nicematrix~can't~be~nested.~However~you~
11323   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11324   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11325   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11326   that~box~in~another~environment~of~nicematrix.
11327 }
11328 \@@_msg_new:nn { Outside~math~mode }
11329 {
11330   Outside~math~mode.\\
11331   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11332   (and~not~in~ \token_to_str:N \vcenter ).
11333 }
11334 \@@_msg_new:nn { One~letter~allowed }
11335 {
11336   Bad~name.\\
11337   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11338   you~have~used~' \l_keys_value_tl '.~Your~key~will~be~ignored.
11339 }
11340 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11341 {
11342   Environment~\{TabularNote\}~forbidden.\\
11343   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11344   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11345   \{TabularNote\}~will~be~ignored.
11346 }
11347 \@@_msg_new:nn { varwidth~not~loaded }
11348 {
11349   varwidth~not~loaded.\\
11350   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11351   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11352   Your~column~will~behave~like~'p'.
11353 }
11354 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11355 {
11356   varwidth~not~loaded.\\
11357   You~can't~use~the~key~'V'~in~your~column~'X'~
11358   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11359   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11360 }

```

```

11361 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11362 {
11363   Unknown~key.\\
11364   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11365   \c_@@_available_keys_str
11366 }
11367 {
11368   The~available~keys~are:~color,~multiplicity,~sep-color,~tikz~
11369   and~total-width~(the~latter~is~meaningful~only~in~cunjunction~with~'tikz').
11370 }

```

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `:`.

```

11371 \@@_msg_new:nnn { Unknown~key~for~Block }
11372 {
11373   Unknown~key. \\
11374   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11375   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11376   \c_@@_available_keys_str
11377 }
11378 {
11379   The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
11380   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11381   opacity,~rounded-corners,~r,~respect-arraystretch,~rules/width,~t,~T,~tikz,~
11382   transparent~and~vlines.
11383 }
11384 \@@_msg_new:nnn { Unknown~key~for~Brace }
11385 {
11386   Unknown~key.\\
11387   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11388   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11389   Your~key~will~be~ignored. \\
11390   \c_@@_available_keys_str
11391 }
11392 {
11393   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
11394   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
11395   right-shorten)~and~yshift.
11396 }
11397 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11398 {
11399   Unknown~key.\\
11400   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11401   \c_@@_available_keys_str
11402 }
11403 {
11404   The~available~keys~are~(in~alphabetic~order):~
11405   delimiters/color,~
11406   rules~(with~the~subkeys~'color'~and~'width'),~
11407   sub-matrix~(several~subkeys)~
11408   and~xdots~(several~subkeys).~
11409   The~latter~is~for~the~command~ \token_to_str:N \line .
11410 }
11411 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11412 {
11413   Unknown~key.\\
11414   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11415   \c_@@_available_keys_str
11416 }
11417 {
11418   The~available~keys~are~(in~alphabetic~order):~
11419   create-cell-nodes,~

```

```

11420     delimiters/color~and~
11421     sub-matrix~(several~subkeys).
11422 }

11423 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11424 {
11425     Unknown~key.\\
11426     The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11427     \c_@@_available_keys_str
11428 }
11429 {
11430     The~available~keys~are~(in~alphabetic~order):~
11431     'delimiters/color',~
11432     'extra-height',~
11433     'hlines',~
11434     'hvlines',~
11435     'left-xshift',~
11436     'name',~
11437     'right-xshift',~
11438     'rules'~(with~the~subkeys~'color'~and~'width'),~
11439     'slim',~
11440     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11441 }

11442 \@@_msg_new:nnn { Unknown~key~for~notes }
11443 {
11444     Unknown~key.\\
11445     The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11446     \c_@@_available_keys_str
11447 }
11448 {
11449     The~available~keys~are~(in~alphabetic~order):~
11450     bottomrule,~
11451     code-after,~
11452     code-before(+),~
11453     detect-duplicates,~
11454     enumitem-keys,~
11455     enumitem-keys-para,~
11456     para,~
11457     label-in-list,~
11458     label-in-tabular~and~
11459     style.
11460 }

11461 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11462 {
11463     Unknown~key.\\
11464     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11465     \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11466     \c_@@_available_keys_str
11467 }
11468 {
11469     The~available~keys~are~(in~alphabetic~order):~
11470     bold,~
11471     cell-space-top-limit(+),~
11472     cell-space-bottom-limit(+),~
11473     cell-space-limits(+),~
11474     color,~
11475     fill~(alias:~rowcolor),~
11476     nb-rows,~
11477     opacity~and~
11478     rounded-corners.
11479 }

11480 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11481 {

```

```

11482   Unknown~key.\\
11483   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11484   \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11485   \c_@@_available_keys_str
11486 }
11487 {
11488   The~available~keys~are~(in~alphabetic~order):~
11489   &-in-blocks,~
11490   allow-duplicate-names,~
11491   ampersand-in-blocks,~
11492   caption-above,~
11493   cell-space-bottom-limit(+),~
11494   cell-space-limits(+),~
11495   cell-space-top-limit(+),~
11496   code-for-first-col(+),~
11497   code-for-first-row(+),~
11498   code-for-last-col(+),~
11499   code-for-last-row(+),~
11500   corners,~
11501   custom-key,~
11502   create-extra-nodes,~
11503   create-medium-nodes,~
11504   create-large-nodes,~
11505   custom-line,~
11506   delimiters~(several~subkeys),~
11507   end-of-row,~
11508   first-col,~
11509   first-row,~
11510   h(v)lines,~
11511   h(v)lines-except-borders,~
11512   last-col,~
11513   last-row,~
11514   left-margin,~
11515   light-syntax,~
11516   light-syntax-expanded,~
11517   matrix/columns-type,~
11518   no-cell-nodes,~
11519   notes~(several~subkeys),~
11520   nullify-dots,~
11521   pgf-node-code,~
11522   renew-dots,~
11523   renew-matrix,~
11524   respect-arraystretch,~
11525   rounded-corners,~
11526   right-margin,~
11527   rules~(with~the~subkeys~'color'~and~'width'),~
11528   small,~
11529   sub-matrix~(several~subkeys),~
11530   vlines,~
11531   xdots~(several~subkeys).
11532 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11533 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11534 {
11535   Unknown~key.\\
11536   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11537   \{NiceArray\}.~Your~key~will~be~ignored. \\
11538   \c_@@_available_keys_str
11539 }
11540 {
11541   The~available~keys~are~(in~alphabetic~order):~
11542   &-in-blocks,~

```

```

11543     ampersand-in-blocks,~
11544     b,~
11545     baseline,~
11546     c,~
11547     cell-space-bottom-limit,~
11548     cell-space-limits,~
11549     cell-space-top-limit,~
11550     code-after,~
11551     code-for-first-col(+),~
11552     code-for-first-row(+),~
11553     code-for-last-col(+),~
11554     code-for-last-row(+),~
11555     columns-width,~
11556     corners,~
11557     create-blocks-in-col,~
11558     create-extra-nodes,~
11559     create-medium-nodes,~
11560     create-large-nodes,~
11561     draw-trees-in-col,~
11562     extra-left-margin,~
11563     extra-right-margin,~
11564     first-col,~
11565     first-row,~
11566     h(v)lines,~
11567     h(v)lines-except-borders,~
11568     last-col,~
11569     last-row,~
11570     left-margin,~
11571     light-syntax,~
11572     light-syntax-expanded,~
11573     name,~
11574     no-cell-nodes,~
11575     nullify-dots,~
11576     pgf-node-code,~
11577     renew-dots,~
11578     respect-arraystretch,~
11579     right-margin,~
11580     rounded-corners,~
11581     rules~(with~the~subkeys~'color'~and~'width'),~
11582     small,~
11583     t,~
11584     vlines,~
11585     xdots/color,~
11586     xdots/shorten-start(+),~
11587     xdots/shorten-end(+),~
11588     xdots/shorten(+)-and~
11589     xdots/line-style.
11590 }

```

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`).

```

11591 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11592 {
11593   Unknown~key.\\
11594   The~key~' \l_keys_key_str '~is~unknown~for~the~
11595   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11596   \c_@@_available_keys_str
11597 }
11598 {
11599   The~available~keys~are~(in~alphabetic~order):~
11600   &~in~blocks,~
11601   ampersand-in-blocks,~
11602   b,~

```

```

11603 baseline,~
11604 c,~
11605 cell-space-bottom-limit,~
11606 cell-space-limits,~
11607 cell-space-top-limit,~
11608 code-after,~
11609 code-for-first-col(+),~
11610 code-for-first-row(+),~
11611 code-for-last-col(+),~
11612 code-for-last-row(+),~
11613 columns-type,~
11614 columns-width,~
11615 corners,~
11616 create-blocks-in-col,~
11617 create-extra-nodes,~
11618 create-medium-nodes,~
11619 create-large-nodes,~
11620 draw-trees-in-col,~
11621 extra-left-margin,~
11622 extra-right-margin,~
11623 first-col,~
11624 first-row,~
11625 h(v)lines,~
11626 h(v)lines-except-borders,~
11627 l,~
11628 last-col,~
11629 last-row,~
11630 left-margin,~
11631 light-syntax,~
11632 light-syntax-expanded,~
11633 name,~
11634 no-cell-nodes,~
11635 nullify-dots,~
11636 pgf-node-code,~
11637 r,~
11638 renew-dots,~
11639 respect-arraystretch,~
11640 right-margin,~
11641 rounded-corners,~
11642 rules~(with~the~subkeys~'color'~and~'width'),~
11643 small,~
11644 t,~
11645 vlines,~
11646 xdots/color,~
11647 xdots/shorten-start(+),~
11648 xdots/shorten-end(+),~
11649 xdots/shorten(+),~and~
11650 xdots/line-style.
11651 }

11652 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11653 {
11654   Unknown~key.\\
11655   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11656   \{NiceTabular\}.~Your~key~will~be~ignored. \\
11657   \c_@@_available_keys_str
11658 }
11659 {
11660   The~available~keys~are~(in~alphabetic~order):~
11661   &~in~blocks,~
11662   ampersand~in~blocks,~
11663   b,~
11664   baseline,~
11665   c,~

```

```

11666 caption,~
11667 cell-space-bottom-limit,~
11668 cell-space-limits,~
11669 cell-space-top-limit,~
11670 code-after,~
11671 code-for-first-col(+),~
11672 code-for-first-row(+),~
11673 code-for-last-col(+),~
11674 code-for-last-row(+),~
11675 columns-width,~
11676 corners,~
11677 custom-line,~
11678 create-blocks-in-col,~
11679 create-extra-nodes,~
11680 create-medium-nodes,~
11681 create-large-nodes,~
11682 draw-trees-in-col,~
11683 extra-left-margin,~
11684 extra-right-margin,~
11685 first-col,~
11686 first-row,~
11687 h(v)lines,~
11688 h(v)lines-except-borders,~
11689 label,~
11690 last-col,~
11691 last-row,~
11692 left-margin,~
11693 light-syntax,~
11694 light-syntax-expanded,~
11695 name,~
11696 no-cell-nodes,~
11697 notes~(several~subkeys),~
11698 nullify-dots,~
11699 pgf-node-code,~
11700 renew-dots,~
11701 respect-arraystretch,~
11702 right-margin,~
11703 rounded-corners,~
11704 rules~(with~the~subkeys~'color'~and~'width'),~
11705 short-caption,~
11706 t,~
11707 tabularnote,~
11708 vlines,~
11709 xdots/color,~
11710 xdots/shorten-start(+),~
11711 xdots/shorten-end(+),~
11712 xdots/shorten(+),~and~
11713 xdots/line-style.
11714 }

11715 \@@_msg_new:nnn { Duplicate-name }
11716 {
11717   Duplicate-name.\
11718   The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11719   the-same-environment-name-twice.~You-can-go-on,~but,~
11720   maybe,~you-will-have-incorrect-results-especially~
11721   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
11722   message-again,~use-the-key~'allow-duplicate-names'~in~
11723   ' \token_to_str:N \NiceMatrixOptions '\
11724   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11725   { For-a-list-of-the-names-already-used,~type-H~<return>. }
11726 }
11727 {
11728   The-names-already-defined-in-this-document-are:-

```

```

11729 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11730 }
11731 \@@_msg_new:nn { caption-above-in-env }
11732 {
11733   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~
11734   Your-key-will-be-ignored.
11735 }
11736 \@@_msg_new:nn { show-cell-names }
11737 {
11738   There-is-no-key~'show-cell-names'~in~nicematrix.\\
11739   You-should-use-the-command~\token_to_str:N \ShowCellNames\
11740   in-the~\token_to_str:N \CodeBefore\ or-the~\token_to_str:N
11741   \CodeAfter.~Your-key-will-be-ignored.
11742 }
11743 \@@_msg_new:nn { Option-auto-for-columns-width }
11744 {
11745   Erroneous-use.\\
11746   You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
11747   Your-key-will-be-ignored.
11748 }
11749 \@@_msg_new:nn { NiceTabularX-without-X }
11750 {
11751   NiceTabularX-without-X.\\
11752   You-should-not-use~\{NiceTabularX\}~without-X-columns.~However,~you-can-go-on.
11753 }
11754 \@@_msg_new:nn { Preamble-forgotten }
11755 {
11756   Preamble-forgotten.\\
11757   You-have-probably-forgotten-the-preamble-of-your~
11758   \@@_full_name_env: .
11759 }
11760 \@@_msg_new:nn { Invalid-col-number }
11761 {
11762   Invalid-column-number.\\
11763   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
11764   specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.~
11765   Maybe-this-is-a-spurious-error-due-to-an-incorrect~'aux'~file.
11766 }
11767 \@@_msg_new:nn { Invalid-row-number }
11768 {
11769   Invalid-row-number.\\
11770   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
11771   specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.~
11772   Maybe-this-is-a-spurious-error-due-to-an-incorrect~'aux'~file.
11773 }
11774 \@@_define_com:NNN p ( )
11775 \@@_define_com:NNN b [ ]
11776 \@@_define_com:NNN v | |
11777 \@@_define_com:NNN V \l \l
11778 \@@_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>	54
10	The environment <code>{NiceArrayWithDelims}</code>	58
11	Construction of the preamble of the array	63
12	The redefinition of <code>\multicolumn</code>	81
13	The environment <code>{NiceMatrix}</code> and its variants	98
	13.1 Definition of <code>{pNiceMatrix}</code>	98
	13.2 The key <code>renew-matrix</code>	99
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	99
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>	125
18	User commands available in the new environments	130
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	137
20	The command <code>\RowStyle</code>	138
21	Colors of cells, rows and columns	141
22	The vertical and horizontal rules	154
23	The empty corners	176
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>	215

30	The keyword <code>\CodeAfter</code>	216
31	The delimiters in the preamble	217
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>	235
37	The key <code>create-blocks-in-col</code>	236
38	The command <code>\ShowCellNames</code>	237
39	We process the options at package loading	239
40	About the package underscore	240
41	Compatibility with <code>threeparttable</code>	241
42	Error messages of the package	241