

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 13, 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 7.10 of `nicematrix`, at the date of 2026/06/13.

```

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 the option `light-syntax-expanded`.

```

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

```

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

```

768 \cs_new_protected:Npn \@@_reset_arraystretch: { \def \arraystretch { 1 } }

```

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

```

769 \bool_new:N \l_@@_auto_columns_width_bool

```

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

```

770 \bool_new:N \g_@@_create_cell_nodes_bool

```

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

```

771 \str_new:N \l_@@_name_str

```

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

```

772 \bool_new:N \l_@@_medium_nodes_bool
773 \bool_new:N \l_@@_large_nodes_bool

```

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

```

774 \bool_new:N \l_@@_except_borders_bool

```

```

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

```

When the key `xdots/nullify` is used, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```

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

```

```

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

```

The options down, up and middle are not documented for the final user because he should use the syntax with \sim , $_$ 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 `~{...}`.

```

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

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, will be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

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

```

```

838 \keys_define:nn { nicematrix / trees }

```

```

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

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

```

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

```

859 \keys_define:nn { nicematrix / Global }
860 {
861   fix-vertex .value_forbidden:n = true ,
862   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
863   brace-shift+ .code:n =
864     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
865   brace-shift+ .value_required:n = true ,
866   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
867   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
868   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
869   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
870   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
871   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
872   &-in-blocks .meta:n = ampersand-in-blocks ,
873   no-cell-nodes .code:n =
874     \bool_set_true:N \l_@@_no_cell_nodes_bool
875     \cs_set_protected:Npn \@@_node_cell:
876       { \set@color \box_use_drop:N \l_@@_cell_box } ,
877   no-cell-nodes .value_forbidden: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.

```

878   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
879   rounded-corners .default:n = 4 pt ,
880   custom-line .code:n = \@@_custom_line:n { #1 } ,
881   default-line .code:n = \@@_default_line:n { #1 } ,
882   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
883   rules .value_required:n = true ,
884   trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
885   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`.

```

886   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
887   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
888   cell-space-top-limit+ .code:n =
889     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,

```

```

890 cell-space-top-limit+ .value_required:n = true ,
891 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
892 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
893 cell-space-bottom-limit+ .code:n =
894   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
895 cell-space-bottom-limit+ .value_required:n = true ,
896 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
897 cell-space-limits .meta:n =
898   {
899     cell-space-top-limit = #1 ,
900     cell-space-bottom-limit = #1 ,
901   } ,
902 cell-space-limits .value_required:n = true ,
903 cell-space-limits+ .meta:n =
904   {
905     cell-space-top-limit += #1 ,
906     cell-space-bottom-limit += #1 ,
907   } ,
908 cell-space-limits+ .value_required:n = true ,
909 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
910 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
911 light-syntax .code:n =
912   \bool_set_true:N \l_@@_light_syntax_bool
913   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
914 light-syntax .value_forbidden:n = true ,
915 light-syntax-expanded .code:n =
916   \bool_set_true:N \l_@@_light_syntax_bool
917   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
918 light-syntax-expanded .value_forbidden:n = true ,

```

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

```

919 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
920 end-of-row .value_required:n = true ,
921 end-of-row .initial:n = ; ,
922
923 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
924 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
925 last-row .int_set:N = \l_@@_last_row_int ,
926 last-row .default:n = -1 ,
927
928 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
929 code-for-first-col .value_required:n = true ,
930 code-for-first-col+ .code:n =
931   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
932 code-for-first-col+ .value_required:n = true ,
933 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
934
935 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
936 code-for-last-col .value_required:n = true ,
937 code-for-last-col+ .code:n =
938   { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
939 code-for-last-col+ .value_required:n = true ,
940 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
941
942 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
943 code-for-first-row .value_required:n = true ,
944 code-for-first-row+ .code:n =
945   { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
946 code-for-first-row+ .value_required:n = true ,
947 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
948
949 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
950 code-for-last-row .value_required:n = true ,

```

```

951 code-for-last-row+ .code:n =
952   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
953 code-for-last-row+ .value_required:n = true ,
954 code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
955
956 hlines .clist_set:N = \l_@@_hlines_clist ,
957 hlines .default:n = all ,
958 vlines .clist_set:N = \l_@@_vlines_clist ,
959 vlines .default:n = all ,
960
961 vl_lines-in-sub-matrix .code:n =
962   {
963     \tl_if_single_token:nTF { #1 }
964     {
965       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
966       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

967     { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
968   }
969   { \@@_error:n { One-letter-allowed } }
970 } ,
971 vl_lines-in-sub-matrix .value_required:n = true ,
972 hv_lines .code:n =
973   {
974     \bool_set_true:N \l_@@_hv_lines_bool
975     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
976     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
977   } ,
978 hv_lines .value_forbidden:n = true ,
979 hv_lines-except-borders .code:n =
980   {
981     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
982     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
983     \bool_set_true:N \l_@@_hv_lines_bool
984     \bool_set_true:N \l_@@_except_borders_bool
985   } ,
986 hv_lines-except-borders .value_forbidden:n = true ,
987 hv_lines-except-borders .code:n =
988   {
989     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
990     \bool_set_true:N \l_@@_hlines_bool
991     \bool_set_true:N \l_@@_except_borders_bool
992   } ,
993 hv_lines-except-borders .value_forbidden:n = true ,
994 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
995 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.

```

996 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
997 renew-dots .value_forbidden:n = true ,
998 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
999 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1000 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1001 create-extra-nodes .meta:n =
1002   { create-medium-nodes , create-large-nodes } ,
1003 left-margin .dim_set:N = \l_@@_left_margin_dim ,
1004 left-margin .default:n = \arraycolsep ,
1005 right-margin .dim_set:N = \l_@@_right_margin_dim ,
1006 right-margin .default:n = \arraycolsep ,
1007 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1008 margin .default:n = \arraycolsep ,

```

```

1009     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1010     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1011     extra-margin .meta:n =
1012       { extra-left-margin = #1 , extra-right-margin = #1 } ,
1013     extra-margin .value_required:n = true ,
1014     respect-arraystretch .code:n =
1015       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1016     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.

```

1017     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1018     pgf-node-code .value_required:n = true ,
1019   }

```

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

```

1020 \keys_define:nn { nicematrix / environments }
1021 {
1022   draw-trees-in-col .code:n =
1023     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1024   draw-trees-in-col .value_required:n = true ,
1025   create-blocks-in-col .code:n =
1026     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1027   create-blocks-in-col .value_required:n = true ,
1028   corners .clist_set:N = \l_@@_corners_clist ,
1029   corners .default:n = { NW , SW , NE , SE } ,
1030   code-before .code:n =
1031     {
1032       \tl_if_empty:nF { #1 }
1033       {
1034         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1035         \bool_set_true:N \l_@@_code_before_bool
1036       }
1037     } ,
1038   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}`.

```

1039   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1040   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1041   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).

```

1042   baseline .tl_set:N = \l_@@_baseline_tl ,
1043   baseline .value_required:n = true ,
1044   baseline .initial:n = c ,
1045   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)`.

```

1046   \str_if_eq:eeTF { #1 } { auto }
1047   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1048   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1049   columns-width .value_required:n = true ,
1050   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.

```

1051   \legacy_if:nF { measuring@ }
1052   {

```



```

1053     \str_set:Ne \l_@@_name_str { #1 }
1054     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1055         { \@@_err_duplicate_names:n { #1 } }
1056         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1057     } ,
1058     name .value_required:n = true ,
1059     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1060     code-after .value_required:n = true ,
1061 }
1062 \cs_set:Npn \@@_err_duplicate_names:n #1
1063 { \@@_error:nn { Duplicate-name } { #1 } }
1064 \keys_define:nn { nicematrix / notes }
1065 {
1066     no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1067     para .bool_set:N = \l_@@_notes_para_bool ,
1068     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1069     code-before .value_required:n = true ,
1070     code-before+ .code:n =
1071         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1072     code-before+ .value_required:n = true ,
1073     code-before~+ .code:n =
1074         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1075     code-before~+ .value_required:n = true ,
1076     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1077     code-after .value_required:n = true ,
1078     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1079     style .cs_set:Np = \@@_notes_style:n #1 ,
1080     style .value_required:n = true ,
1081     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1082     label-in-tabular .value_required:n = true ,
1083     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1084     label-in-list .value_required:n = true ,
1085     enumitem-keys .code:n =
1086     {
1087         \AtBeginDocument
1088         {
1089             \IfPackageLoadedT { enumitem }
1090             { \setlist* [ tabularnotes ] { #1 } }
1091         }
1092     } ,
1093     enumitem-keys .value_required:n = true ,
1094     enumitem-keys-para .code:n =
1095     {
1096         \AtBeginDocument
1097         {
1098             \IfPackageLoadedT { enumitem }
1099             { \setlist* [ tabularnotes* ] { #1 } }
1100         }
1101     } ,
1102     enumitem-keys-para .value_required:n = true ,
1103     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1104     unknown .code:n =
1105         \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1106 }

```

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

```

1107 \keys_define:nn { nicematrix / delimiters }
1108 {
1109   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1110   color .tl_set:N = \l_@@_delimiters_color_tl ,
1111   color .value_required:n = true ,
1112 }

```

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

```

1113 \keys_define:nn { nicematrix }
1114 {
1115   NiceMatrixOptions .inherit:n =
1116     { nicematrix / Global } ,
1117   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1118   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1119   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1120   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1121   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1122   SubMatrix / rules .inherit:n = nicematrix / rules ,
1123   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1124   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1125   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1126   NiceMatrix .inherit:n =
1127     {
1128       nicematrix / Global ,
1129       nicematrix / environments ,
1130     } ,
1131   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1132   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1133   NiceMatrix / trees .inherit:n = nicematrix / trees ,
1134   NiceTabular .inherit:n =
1135     {
1136       nicematrix / Global ,
1137       nicematrix / environments
1138     } ,
1139   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1140   NiceTabular / rules .inherit:n = nicematrix / rules ,
1141   NiceTabular / notes .inherit:n = nicematrix / notes ,
1142   NiceTabular / trees .inherit:n = nicematrix / trees ,
1143   NiceArray .inherit:n =
1144     {
1145       nicematrix / Global ,
1146       nicematrix / environments ,
1147     } ,
1148   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1149   NiceArray / rules .inherit:n = nicematrix / rules ,
1150   NiceArray / trees .inherit:n = nicematrix / trees ,
1151   pNiceArray .inherit:n =
1152     {
1153       nicematrix / Global ,
1154       nicematrix / environments ,
1155     } ,
1156   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1157   pNiceArray / rules .inherit:n = nicematrix / rules ,
1158   pNiceArray / trees .inherit:n = nicematrix / trees
1159 }

```

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

```

1160 \keys_define:nn { nicematrix / NiceMatrixOptions }
1161 {
1162   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

1163 delimiters / color .value_required:n = true ,
1164 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1165 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1166 delimiters .value_required:n = true ,
1167 width .dim_set:N = \l_@@_width_dim ,
1168 last-col .code:n =
1169   \tl_if_empty:nF { #1 }
1170     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1171     \int_zero:N \l_@@_last_col_int ,
1172 small .bool_set:N = \l_@@_small_bool ,
1173 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.

```

1174 renew-matrix .code:n = \@@_renew_matrix: ,
1175 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}`.

```

1176 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.

```

1177 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)`.

```

1178   \str_if_eq:eeTF { #1 } { auto }
1179   { \@@_error:n { Option-auto-for-columns-width } }
1180   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1181 allow-duplicate-names .code:n =
1182   \cs_set:Nn \@@_err_duplicate_names:n { } ,
1183 allow-duplicate-names .value_forbidden:n = true ,
1184 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1185 notes .value_required:n = true ,
1186 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1187 sub-matrix .value_required:n = true ,
1188 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1189 matrix / columns-type .value_required:n = true ,
1190 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1191 unknown .code:n =
1192   \@@_unknown_key:nn
1193     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1194     { Unknown-key-for-NiceMatrixOptions }
1195 }

```

`\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.

```

1196 \NewDocumentCommand \NiceMatrixOptions { m }
1197 { \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.

```

1198 \keys_define:nn { nicematrix / NiceMatrix }
1199 {
1200   last-col .code:n = \tl_if_empty:nTF { #1 }

```

```

1201         {
1202             \bool_set_true:N \l_@@_last_col_without_value_bool
1203             \int_set:Nn \l_@@_last_col_int { -1 }
1204         }
1205         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1206     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1207     columns-type .value_required:n = true ,
1208     l .meta:n = { columns-type = l } ,
1209     r .meta:n = { columns-type = r } ,
1210     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1211     delimiters / color .value_required:n = true ,
1212     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1213     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1214     delimiters .value_required:n = true ,
1215     small .bool_set:N = \l_@@_small_bool ,
1216     small .value_forbidden:n = true ,
1217     unknown .code:n =
1218         \@@_unknown_key:nn
1219             { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1220             { Unknown-key-for-NiceMatrix }
1221 }

```

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

```

1222 \keys_define:nn { nicematrix / NiceArray }
1223 {

```

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.

```

1224     small .bool_set:N = \l_@@_small_bool ,
1225     small .value_forbidden:n = true ,
1226     last-col .code:n = \tl_if_empty:nF { #1 }
1227         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1228         \int_zero:N \l_@@_last_col_int ,
1229     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1230     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1231     unknown .code:n =
1232         \@@_unknown_key:nn
1233             { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1234             { Unknown-key-for-NiceArray }
1235 }
1236 \keys_define:nn { nicematrix / pNiceArray }
1237 {
1238     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1239     last-col .code:n = \tl_if_empty:nF { #1 }
1240         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1241         \int_zero:N \l_@@_last_col_int ,
1242     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1243     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1244     delimiters / color .value_required:n = true ,
1245     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1246     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1247     delimiters .value_required:n = true ,
1248     small .bool_set:N = \l_@@_small_bool ,
1249     small .value_forbidden:n = true ,
1250     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1251     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1252     unknown .code:n =
1253         \@@_unknown_key:nn
1254             { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1255             { Unknown-key-for-NiceMatrix }
1256 }

```

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

```
1257 \keys_define:nn { nicematrix / NiceTabular }
1258 {
```

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.

```
1259   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1260           \bool_set_true:N \l_@@_width_used_bool ,
1261   width .value_required:n = true ,
1262   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1263   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1264   tabularnote .value_required:n = true ,
1265   caption .tl_set:N = \l_@@_caption_tl ,
1266   caption .value_required:n = true ,
1267   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1268   short-caption .value_required:n = true ,
1269   label .tl_set:N = \l_@@_label_tl ,
1270   label .value_required:n = true ,
1271   last-col .code:n = \tl_if_empty:nF { #1 }
1272           { \@@_error:n { last-col-non-empty-for-NiceArray } }
1273           \int_zero:N \l_@@_last_col_int ,
1274   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1275   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1276   unknown .code:n =
1277     \@@_unknown_key:nn
1278     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1279     { Unknown-key-for-NiceTabular }
1280 }
```

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
```

```
1281 \keys_define:nn { nicematrix / CodeAfter }
1282 {
1283   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1284   delimiters / color .value_required:n = true ,
1285   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1286   rules .value_required:n = true ,
1287   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1288   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1289   sub-matrix .value_required:n = true ,
1290   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1291 }
```

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}`).

```
1292 \cs_new_protected:Npn \@@_cell_begin:
1293 {
```

`\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.

```
1294 \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).

```
1295 \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.

```
1296 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1297 \int_gincr:N \c_jCol
```

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

Here is a version with the standard syntax of L3.

```
\int_compare:nNnT { \c_jCol } = { 1 }
{ \int_compare:nNnT \l_@@_first_col_int = { 1 } { \@@_begin_of_row: } }
```

We will use a version a little more efficient.

```
1298 \if_int_compare:w \c_jCol = \c_one_int
1299 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1300 \@@_begin_of_row:
1301 \fi:
1302 \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:`.

```
1303 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1304 \@@_tuning_not_tabular_begin:
1305 \@@_tuning_exterior_rows:
1306 \g_@@_row_style_tl
1307 }
1308 \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.

```
1309 \cs_new_protected:Npn \@@_tuning_first_last_row:
1310 {
1311 \if_int_compare:w \c_iRow = \c_zero_int
1312 \if_int_compare:w \c_jCol > \c_zero_int
1313 \l_@@_code_for_first_row_tl
1314 \@@_tuning_key_small: % 2026/04/06
1315 \xglobal \colorlet { nicematrix-first-row } { . }
1316 \fi:
```

```

1317     \else:
1318         \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1319     \fi:
1320 }

```

The following command will be nullified unless there is a last row and we know its value (*ie*: $\l_1\@@_lat_row_int > 0$).

```

\cs_new_protected:Npn \@@_tuning_last_row:
{
    \int_compare:nNnT { \c@iRow } = { \l_1\@@_last_row_int }
    {
        \l_1\@@_code_for_last_row_tl
        \xglobal \colorlet { nicematrix-last-row } { . }
    }
}

```

We will use a version a little more efficient.

```

1321 \cs_new_protected:Npn \@@_tuning_last_row:
1322 {
1323     \if_int_compare:w \c@iRow = \l_1\@@_last_row_int
1324         \l_1\@@_code_for_last_row_tl
1325         \@@_tuning_key_small: % 2026/04/06
1326         \xglobal \colorlet { nicematrix-last-row } { . }
1327     \fi:
1328 }

```

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

```

1329 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1330 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1331 {
1332     \m@th
1333     $ % $

```

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

```

1334     \@@_tuning_key_small:
1335 }
1336 \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.

```

1337 \cs_new_protected:Npn \@@_begin_of_row:
1338 {
1339     \int_gincr:N \c@iRow
1340     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1341     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1342     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1343     \pgfpicture
1344     \pgfrememberpicturepositiononpagetrue
1345     \pgfcoordinate
1346     { \@@_env: - row - \int_use:N \c@iRow - base }
1347     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1348     \str_if_empty:NF \l_@@_name_str
1349     {
1350         \pgfnodelalias
1351         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1352         { \@@_env: - row - \int_use:N \c@iRow - base }
1353     }
1354     \endpgfpicture
1355 }

```

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.

```
1356 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1357 {
1358   \if_int_compare:w \c@iRow = \c_zero_int
1359     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1360       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1361     \fi:
1362     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1363       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1364     \fi:
1365   \else:
1366     \if_int_compare:w \c@iRow = \c_one_int
1367       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1368         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1369       \fi:
1370     \fi:
1371   \fi:
1372 }

1373 \cs_new_protected:Npn \@@_rotate_cell_box:
1374 {
1375   \box_rotate:Nn \l_@@_cell_box
1376   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1377   \bool_if:NTF \g_@@_rotate_c_bool
1378   {
1379     \hbox_set:Nn \l_@@_cell_box
1380     {
1381       \IfFormatAtLeastTF { 2026-04-01 }
1382       { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1383       {
1384         \m@th
1385         $ % $
1386         \vcenter { \box_use:N \l_@@_cell_box }
```



```

1387         $ % $
1388     }
1389 }
1390 }
1391 {
1392     \int_compare:nNtT \c@iRow = \l_@@_last_row_int
1393     {
1394         \vbox_set_top:Nn \l_@@_cell_box
1395         {
1396             \vbox_to_zero:n { }
1397             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1398             \box_use:N \l_@@_cell_box
1399         }
1400     }
1401 }
1402 \bool_gset_false:N \g_@@_rotate_bool
1403 \bool_gset_false:N \g_@@_rotate_c_bool
1404 \bool_gset_false:N \g_@@_rotate_minus_bool
1405 }

```

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:nNtT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
    {
        \dim_compare:nNtT \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:nNtT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
    {
        \dim_compare:nNtT \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:nNtT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
    {
        \dim_compare:nNtT \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.

```

1406 \cs_set_protected:Npn \@@_adjust_size_box:
1407 {
1408     \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1409     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1410     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1411     \fi:
1412     \global \g_@@_blocks_wd_dim = \c_zero_dim
1413     \fi:
1414     \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1415     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1416     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1417     \fi:
1418     \global \g_@@_blocks_dp_dim = \c_zero_dim
1419     \fi:
1420     \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1421     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1422     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1423     \fi:

```

```

1424     \global \g_@@_blocks_ht_dim = \c_zero_dim
1425     \fi:
1426 }

1427 \cs_new_protected:Npn \@@_cell_end:
1428 {

```

The following command is nullified in the tabulars.

```

1429     \@@_tuning_not_tabular_end:
1430     \hbox_set_end:
1431     \@@_cell_end_i:
1432 }

```

```

\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.

```

1433 \cs_new_protected:Npn \@@_cell_end_i:
1434 {

```

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.

```

1435     \g_@@_cell_after_hook_tl
1436     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1437     \@@_adjust_size_box:

1438     \box_set_ht:Nn \l_@@_cell_box
1439     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1440     \box_set_dp:Nn \l_@@_cell_box
1441     { \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”).

```

1442     \@@_update_max_cell_width:

```

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

```

1443     \@@_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.

```

1444 \bool_if:NTF \g_@@_empty_cell_bool
1445 { \box_use_drop:N \l_@@_cell_box }
1446 {
1447   \bool_if:NTF \g_@@_not_empty_cell_bool
1448   \@@_print_node_cell:
1449   {
1450     \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1451     \@@_print_node_cell:
1452     \else:
1453       \box_use_drop:N \l_@@_cell_box
1454     \fi:
1455   }
1456 }
1457 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1458 \global \g_@@_col_total_int = \c@jCol
1459 \fi:
1460 \global \let \g_@@_empty_cell_bool \c_false_bool
1461 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1462 }

```

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:

```

1463 \cs_new_protected:Npn \@@_update_max_cell_width:
1464 {
1465   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1466   \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1467   \fi:
1468 }

```

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`).

```

1469 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1470 {
1471   \@@_math_toggle:
1472   \hbox_set_end:

```

```

1473 \bool_if:NF \g_@@_rotate_bool
1474 {
1475   \hbox_set:Nn \l_@@_cell_box
1476   {
1477     \makebox [ \l_@@_col_width_dim ] [ s ]
1478     { \hbox_unpack_drop:N \l_@@_cell_box }
1479   }
1480 }
1481 \@@_cell_end_i:
1482 }

```

```

1483 \pgfset
1484 {
1485   nicematrix / cell-node /.style =
1486   {
1487     inner~sep = \c_zero_dim ,
1488     minimum~width = \c_zero_dim
1489   }
1490 }

```

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.

```

1491 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1492 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1493 {
1494   \use:c
1495   {
1496     __siunitx_table_align_
1497     \bool_if:NTF \l__siunitx_table_text_bool
1498     \l__siunitx_table_align_text_tl
1499     \l__siunitx_table_align_number_str
1500     :n
1501   }
1502   { #1 }
1503 }

```

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!

```

1504 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1505 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1506 {
1507   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1508   \hbox:n
1509   {
1510     \pgfsys@markposition
1511     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1512   }
1513   #1
1514   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1515   \hbox:n
1516   {
1517     \pgfsys@markposition
1518     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1519   }
1520 }

```

```

1521 \cs_new_protected:Npn \@@_print_node_cell:

```

```

1522 {
1523   \socket_use:nn { nicematrix / siunitx-wrap }
1524   { \socket_use:nn { nicematrix / create-cell-nodes } { \l_@@_node_cell: } }
1525 }

```

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

```

1526 \cs_new_protected:Npn \l_@@_node_cell:
1527 {
1528   \pgfpicture
1529   \pgfsetbaseline \c_zero_dim
1530   \pgfrememberpicturepositiononpagetrue
1531   \pgfset { nicematrix / cell-node }
1532   \pgfnode
1533   { rectangle }
1534   { base }
1535   {

```

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).

```

1536   \sys_if_engine_xetex:T { \set@color }
1537   \box_use:N \l_@@_cell_box
1538 }
1539 { \l_@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1540 { \l_@@_pgf_node_code_tl }
1541 \str_if_empty:NF \l_@@_name_str
1542 {
1543   \pgfnodealias
1544   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1545   { \l_@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1546 }
1547 \endpgfpicture
1548 }

```

The second argument of the following command `\l_@@_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:

```

\l_@@_draw_Cdots:nnn {2}{2}{}
\l_@@_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).

```

1549 \cs_new_protected:Npn \l_@@_instruction_of_type:nnn #1 #2 #3
1550 {
1551   \bool_if:nTF { #1 } { \tl_gput_left:ce \tl_gput_right:ce
1552     { \g_@@_#2 _ lines _ tl }
1553     {
1554       \use:c { @ _ draw _ #2 : nnn }
1555       { \int_use:N \c@iRow }
1556       { \int_use:N \c@jCol }

```

```

1557         { \exp_not:n { #3 } }
1558     }
1559 }

```

```

1560 \cs_new_protected:Npn \@@_array:n
1561 {
1562     \dim_set:Nn \col@sep
1563     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1564     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1565     { \def \@halignto { } }
1566     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1567     \@tabarray

```

\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose the \array (of array) with the option t and the right translation will be done further. Remark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here. \str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

```

1568     [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1569 }
1570 \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.

```

1571 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign

```

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

```

1572 \cs_new_protected:Npn \@@_create_row_node:
1573 {
1574     \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1575     {
1576         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1577         \@@_create_row_node_i:
1578     }
1579 }
1580 \cs_new_protected:Npn \@@_create_row_node_i:
1581 {

```

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

```

1582     \hbox
1583     {
1584         \bool_if:NT \l_@@_code_before_bool
1585         {
1586             \vtop
1587             {
1588                 \skip_vertical:N 0.5\arrayrulewidth
1589                 \pgfsys@markposition
1590                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1591                 \skip_vertical:N -0.5\arrayrulewidth
1592             }
1593         }
1594         \pgfpicture
1595         \pgfrememberpicturepositiononpagetrue
1596         \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1597         { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1598         \str_if_empty:NF \l_@@_name_str
1599         {
1600             \pgfnodealias
1601             { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1602             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }

```

```

1603     }
1604     \endpgfpicture
1605 }
1606 }

1607 \cs_new_protected:Npn \@@_in_everycr:
1608 {
1609     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1610     \tbl_update_cell_data_for_next_row:
1611     \int_gzero:N \c@jCol
1612     \bool_gset_false:N \g_@@_after_col_zero_bool
1613     \bool_if:NF \g_@@_row_of_col_done_bool
1614     {
1615         \@@_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).

```

1616     \clist_if_empty:NF \l_@@_hlines_clist
1617     {
1618         \str_if_eq:eeF \l_@@_hlines_clist { all }
1619         {
1620             \clist_if_in:NiT
1621             \l_@@_hlines_clist
1622             { \int_eval:n { \c@iRow + 1 } }
1623         }
1624     }

```

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.

```

1625         \int_compare:nNnT \c@iRow > { -1 }
1626         {
1627             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1628             { \hrule height \arrayrulewidth width \c_zero_dim }
1629         }
1630     }
1631 }
1632 }
1633 }

```

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

```

1634 \cs_set_protected:Npn \@@_renew_dots:
1635 {
1636     \cs_set_eq:NN \ldots \@@_Ldots:
1637     \cs_set_eq:NN \cdots \@@_Cdots:
1638     \cs_set_eq:NN \vdots \@@_Vdots:
1639     \cs_set_eq:NN \ddots \@@_Ddots:
1640     \cs_set_eq:NN \iddots \@@_Iddots:
1641     \cs_set_eq:NN \dots \@@_Ldots:
1642     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1643 }

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁵.

```

1644 \AtBeginDocument

```

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

```

1645 {
1646   \IfPackageLoadedTF { booktabs }
1647   {
1648     \cs_new_protected:Npn \@@_patch_booktabs:
1649       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1650   }
1651   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1652 }

```

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`.

```

1653 \cs_new_protected:Npn \@@_some_initialization:
1654 {
1655   \@@_everycr:
1656   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1657   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1658   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1659   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1660   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1661   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1662 }

```

`\@@_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`.

```

1663 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1664 {

```

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

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

```

1665   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1666   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1667   \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.

```

1668   \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.

```

1669   \fp_gzero:N \g_@@_total_X_weight_fp
1670   \bool_gset_false:N \g_@@_V_of_X_bool
1671   \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1672   \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

```

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


```

1673 \@@_patch_booktabs:
1674 \box_clear_new:N \l_@@_cell_box
1675 \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}`).

```

1676 \bool_if:NT \l_@@_small_bool
1677 {
1678     \def \arraystretch { 0.47 }
1679     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1680     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1681 }

```

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!

```

1682 \bool_if:NT \g_@@_create_cell_nodes_bool
1683 {
1684     \tl_put_right:Nn \@@_begin_of_row:
1685     {
1686         \pgfsys@markposition
1687         { \@@_env: - row - \int_use:N \c@iRow - base }
1688     }
1689     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1690 }

```

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`.

```

1691 \def \ar@ialign
1692 {
1693     \tbl_init_cell_data_for_table:
1694     \@@_some_initialization:
1695     \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`.

```

1696     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1697     \halign
1698 }

```

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).

```

1699 \cs_set_eq:NN \@@_old_ldots: \ldots
1700 \cs_set_eq:NN \@@_old_cdots: \cdots
1701 \cs_set_eq:NN \@@_old_vdots: \vdots
1702 \cs_set_eq:NN \@@_old_ddots: \ddots
1703 \cs_set_eq:NN \@@_old_iddots: \iddots
1704 \bool_if:NTF \l_@@_standard_cline_bool
1705 { \cs_set_eq:NN \cline \@@_standard_cline: }
1706 { \cs_set_eq:NN \cline \@@_cline: }
1707 \cs_set_eq:NN \Ldots \@@_Ldots:
1708 \cs_set_eq:NN \Cdots \@@_Cdots:
1709 \cs_set_eq:NN \Vdots \@@_Vdots:
1710 \cs_set_eq:NN \Ddots \@@_Ddots:
1711 \cs_set_eq:NN \Iddots \@@_Iddots:

```

```

1712 \cs_set_eq:NN \Hline \@@_Hline:
1713 \cs_set_eq:NN \Hspace \@@_Hspace:
1714 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1715 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1716 \cs_set_eq:NN \Block \@@_Block:
1717 \cs_set_eq:NN \rotate \@@_rotate:
1718 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1719 \cs_set_eq:NN \dotfill \@@_dotfill:
1720 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1721 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1722 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1723 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1724 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1725 \cs_set_eq:NN \TopRule \@@_TopRule
1726 \cs_set_eq:NN \MidRule \@@_MidRule
1727 \cs_set_eq:NN \BottomRule \@@_BottomRule
1728 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1729 \cs_set_eq:NN \Hbrace \@@_Hbrace
1730 \cs_set_eq:NN \Vbrace \@@_Vbrace
1731 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1732 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1733 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1734 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1735 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1736 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1737 \int_if_zero:nTF \l_@@_first_row_int
1738 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1739 {
1740 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1741 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1742 }
1743 \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:`.

```

1744 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1745 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1746 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1747 \@@_revert_colortbl:

```

If there is one or several commands `\tablarnote` 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!).

```

1748 \tl_if_exist:NT \l_@@_note_in_caption_tl
1749 {
1750 \tl_if_empty:NF \l_@@_note_in_caption_tl
1751 {
1752 \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1753 \int_gset:Nn \c@tablarnote \l_@@_note_in_caption_tl
1754 }
1755 }

```

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).

```

1756 \seq_gclear:N \g_@@_multicolumn_cells_seq
1757 \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`).

```
1758 \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).

```
1759 \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`).

```
1760 \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.

```
1761 \int_gzero:N \g_@@_col_total_int
```

```
1762 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1763 \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.

```
1764 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1765 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1766 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1767 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1768 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1769 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl
```

```
1770 \tl_gclear:N \g_nicematrix_code_before_tl
```

```
1771 \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]`).

```
1772 \dim_zero_new:N \l_@@_left_delim_dim
1773 \dim_zero_new:N \l_@@_right_delim_dim
1774 \bool_if:NTF \g_@@_delims_bool
1775 {
```

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

```
1776 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1777 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1778 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1779 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1780 }
1781 {
1782 \dim_gset:Nn \l_@@_left_delim_dim
1783 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1784 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1785 }
1786 }
```

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.

```
1787 \cs_new_protected:Npn \@@_pre_array:
1788 {
1789 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1790 \int_gzero_new:N \c@iRow
1791 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1792 \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).

```

1793 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1794 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1795 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1796 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1797 \bool_if:NT \g_@@_aux_found_bool
1798 {
1799   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1800   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1801   \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1802   \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1803 }

```

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).

```

1804 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1805 {
1806   \bool_set_true:N \l_@@_last_row_without_value_bool
1807   \bool_if:NT \g_@@_aux_found_bool
1808   { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1809 }
1810 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1811 {
1812   \bool_if:NT \g_@@_aux_found_bool
1813   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1814 }

```

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”.

```

1815 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1816 {
1817   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1818   {
1819     \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1820     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1821     \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1822     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1823   }
1824 }

1825 \seq_gclear:N \g_@@_cols_vlism_seq
1826 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1827 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1828 \@@_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).

```

1829 \hbox_set:Nw \l_@@_the_array_box
1830 \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1831 \skip_horizontal:N \l_@@_extra_left_margin_dim
1832 \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).

```

1833 \m@th
1834 $ % $
1835 \bool_if:NTF \l_@@_light_syntax_bool
1836 { \use:c { @@-light-syntax } }
1837 { \use:c { @@-normal-syntax } }
1838 }

```

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

```

1839 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1840 {
1841   \tl_set:Nn \l_tmpa_tl { #1 }
1842   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1843   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1844   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1845   \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...

```

1846 \@@_pre_array:
1847 }

```

9 The `\CodeBefore`

```

1848 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1849 \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).

```

1850 \cs_new_protected:Npn \@@_pre_code_before:
1851 {

```

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.

```

1852 \pgfsys@markposition { \@@_env: - position }
1853 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1854 \pgfpicture
1855 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1856 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1857 {
1858   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1859   \pgfcoordinate { \@@_env: - row - ##1 }
1860   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1861 }

```

Now, the recreation of the `col` nodes.

```

1862 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1863 {

```

```

1864     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1865     \pgfcoordinate { \@@_env: - col - ##1 }
1866     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1867 }

```

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

```

1868     \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1869     \endpgfpicture

```

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

```

1870     \@@_create_diag_nodes:

```

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

```

1871     \@@_create_blocks_nodes:
1872     \IfPackageLoadedT { tikz }
1873     {
1874         \tikzset
1875         {
1876             every-picture / .style =
1877             { overlay , name-prefix = \@@_env: - }
1878         }
1879     }
1880     \cs_set_eq:NN \cellcolor \@@_cellcolor
1881     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1882     \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1883     \cs_set_eq:NN \rowcolor \@@_rowcolor
1884     \cs_set_eq:NN \rowcolors \@@_rowcolors
1885     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1886     \cs_set_eq:NN \arraycolor \@@_arraycolor
1887     \cs_set_eq:NN \columncolor \@@_columncolor
1888     \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1889     \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1890     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1891     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1892     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1893     \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1894     \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1895 }

1896 \cs_new_protected:Npn \@@_exec_code_before:
1897 {

```

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...

```

1898     \clist_map_inline:Nn \l_@@_corners_cells_clist
1899     { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1900     \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`.

```

1901     \@@_add_to_colors_seq:nn { { nocolor } } { }
1902     \bool_gset_false:N \g_@@_create_cell_nodes_bool
1903     \group_begin:

```

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

```

1904     \if_mode_math:
1905         \@@_exec_code_before_i:
1906     \else:
1907         $ % $
1908         \@@_exec_code_before_i:
1909         $ % $
1910     \fi:

```

```

1911 \group_end:
1912 }

```

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`).

```

1913 \cs_new_protected:Npn \@@_exec_code_before_i:
1914 {
1915   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1916   { \@@_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`.

```

1917   \exp_last_unbraced:No \@@_CodeBefore_keys:
1918   \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.

```

1919   \@@_actually_color:
1920   \l_@@_code_before_tl
1921   \q_stop
1922 }

```

```

1923 \keys_define:nn { nicematrix / CodeBefore }
1924 {
1925   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1926   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1927   sub-matrix .value_required:n = true ,
1928   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1929   delimiters / color .value_required:n = true ,
1930   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1931 }
1932 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1933 {
1934   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1935   \@@_CodeBefore:w
1936 }

```

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.

```

1937 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1938 {
1939   \bool_if:NTF \g_@@_aux_found_bool
1940   {
1941     \@@_pre_code_before:
1942     \legacy_if:nF { measuring@ } { #1 }
1943   }

```

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).

```

1944   {
1945     \pgfsys@markposition { \@@_env: - position }
1946     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1947     \pgfpicture
1948     \pgf@relevantforpicturesizefalse
1949     \endpgfpicture

```

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

```

1950 \pgfpicture
1951 \pgfrememberpicturerepositiononpagetrue
1952 \endpgfpicture

```

The following picture corresponds to \@@_create_blocks_nodes:.

```

1953 \pgfpicture
1954 \pgf@relevantforpicturesizefalse
1955 \pgfrememberpicturerepositiononpagetrue
1956 \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

1957 \pgfpicture
1958 \pgf@relevantforpicturesizefalse
1959 \endpgfpicture
1960 }
1961 }

```

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.

```

1962 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1963 {
1964   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1965   {
1966     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1967     \pgfcoordinate { \@@_env: - row - ##1 - base }
1968     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1969     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1970     {
1971       \cs_if_exist:cT
1972       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1973       {
1974         \pgfsys@getposition
1975         { \@@_env: - ##1 - #####1 - NW }
1976         \@@_node_position:
1977         \pgfsys@getposition
1978         { \@@_env: - ##1 - #####1 - SE }
1979         \@@_node_position_i:
1980         \@@_pgf_rect_node:nnn
1981         { \@@_env: - ##1 - #####1 }
1982         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1983         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1984       }
1985     }
1986   }
1987   \@@_create_extra_nodes:
1988   \@@_create_aliases_last:
1989 }

```

```

1990 \cs_new_protected:Npn \@@_create_aliases_last:
1991 {
1992   \int_step_inline:nn \c@iRow
1993   {
1994     \pgfnodealias
1995     { \@@_env: - ##1 - last }
1996     { \@@_env: - ##1 - \int_use:N \c@jCol }
1997   }
1998   \int_step_inline:nn \c@jCol
1999   {
2000     \pgfnodealias
2001     { \@@_env: - last - ##1 }
2002     { \@@_env: - \int_use:N \c@iRow - ##1 }
2003   }

```



```

2004 \pgfnodealias
2005 { \@@_env: - last - last }
2006 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2007 }

```

```

2008 \cs_new_protected:Npn \@@_create_blocks_nodes:
2009 {
2010   \pgfpicture
2011   \pgf@relevantforpicturesizefalse
2012   \pgfrememberpicturepositiononpagetrue
2013   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2014   { \@@_create_one_block_node:nnnnn ##1 }
2015   \endpgfpicture
2016 }

```

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

```

2017 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2018 {
2019   \tl_if_empty:nF { #5 }
2020   {
2021     \@@_qpoint:n { col - #2 }
2022     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2023     \@@_qpoint:n { #1 }
2024     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2025     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2026     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2027     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2028     \@@_pgf_rect_node:nnnnn
2029     { \@@_env: - #5 }
2030     { \dim_use:N \l_tmpa_dim }
2031     { \dim_use:N \l_tmpb_dim }
2032     { \dim_use:N \l_@@_tmpc_dim }
2033     { \dim_use:N \pgf@y }
2034   }
2035 }

```

10 The environment {NiceArrayWithDelims}

```

2036 \NewDocumentEnvironment { NiceArrayWithDelims }
2037 { m m O { } m ! O { } t \CodeBefore }
2038 {
2039   \@@_provide_pgfsyspdfmark:
2040   \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

2041   \bgroup

2042   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2043   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2044   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2045   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2046   \int_gzero:N \g_@@_block_box_int
2047   \dim_gzero:N \g_@@_width_last_col_dim
2048   \dim_gzero:N \g_@@_width_first_col_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).

```

2049 \bool_gset_false:N \g_@@_row_of_col_done_bool
2050 \str_if_empty:NT \g_@@_name_env_str
2051 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2052 \bool_if:NTF \l_@@_tabular_bool
2053 \mode_leave_vertical:
2054 \@@_test_if_math_mode:
2055 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2056 \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.

```

2057 \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.

```

2058 \cs_if_exist:NT \tikz@library@external@loaded
2059 {
2060 \tikzexternaldisable
2061 \cs_if_exist:NT \ifstandalone
2062 { \tikzset { external / optimize = false } }
2063 }

```

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

```

2064 \int_gincr:N \g_@@_env_int
2065 \bool_if:NF \l_@@_block_auto_columns_width_bool
2066 { \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.

```

2067 \seq_gclear:N \g_@@_blocks_seq
2068 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

2069 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2070 \seq_gclear:N \g_@@_pos_of_xdots_seq
2071 \tl_gclear_new:N \g_@@_code_before_tl
2072 \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.

```

2073 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2074 {
2075 \bool_gset_true:N \g_@@_aux_found_bool
2076 \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2077 }
2078 { \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.

```

2079 \tl_gclear:N \g_@@_aux_tl
2080 \tl_if_empty:NF \g_@@_code_before_tl
2081 {
2082 \bool_set_true:N \l_@@_code_before_bool
2083 \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2084 }
2085 \tl_if_empty:NF \g_@@_pre_code_before_tl
2086 { \bool_set_true:N \l_@@_code_before_bool }

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

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

```

2087 \bool_if:NTF \g_@@_delims_bool
2088 { \keys_set:nn { nicematrix / pNiceArray } }
2089 { \keys_set:nn { nicematrix / NiceArray } }
2090 { #3 , #5 }

2091 \@@_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:.

```

2092 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2093 }

```

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

```

2094 {
2095   \bool_if:NTF \l_@@_light_syntax_bool
2096   { \use:c { end @@-light-syntax } }
2097   { \use:c { end @@-normal-syntax } }
2098   $ % $
2099   \skip_horizontal:N \l_@@_right_margin_dim
2100   \skip_horizontal:N \l_@@_extra_right_margin_dim
2101   \hbox_set_end:
2102   \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.

```

2103 \bool_if:NT \l_@@_width_used_bool
2104 {
2105   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2106   { \@@_error_or_warning:n { width-without-X-columns } }
2107 }

```

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 .

```

2108 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2109 \@@_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).

```

2110 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2111 {
2112   \bool_if:NF \l_@@_last_row_without_value_bool
2113   {
2114     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2115     {
2116       \@@_error:n { Wrong-last-row }
2117       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2118     }
2119   }
2120 }

```

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

```

2121 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2122 \bool_if:NTF \g_@@_last_col_found_bool
2123 { \int_gdecr:N \c@jCol }
2124 {
2125   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2126   { \@@_error:n { last~col~not~used } }
2127 }

```

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

```

2128 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2129 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2130 { \int_gdecr:N \c@iRow }

```

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

```

2131 \int_if_zero:nT \l_@@_first_col_int
2132 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2133 \bool_if:nTF { ! \g_@@_delims_bool }
2134 {
2135   \str_if_eq:eeTF c \l_@@_baseline_tl
2136   \@@_use_arraybox_with_notes_c:
2137   {
2138     \str_if_eq:eeTF b \l_@@_baseline_tl
2139     \@@_use_arraybox_with_notes_b:
2140     \@@_use_arraybox_with_notes:
2141   }
2142 }

```

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).

```

2143 {
2144   \int_if_zero:nTF \l_@@_first_row_int
2145   {
2146     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2147     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2148   }
2149   { \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

```

2150 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2151 {
2152   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2153   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2154 }
2155 { \dim_zero:N \l_tmpb_dim }
2156 \hbox_set:Nn \l_tmpa_box
2157 {
2158   \m@th
2159   $ % $
2160   \@@_color:o \l_@@_delimiters_color_tl
2161   \exp_after:wN \left \g_@@_left_delim_tl
2162   \vcenter
2163   {

```

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

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

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```

2164         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2165         \hbox
2166         {
2167             \bool_if:NTF \l_@@_tabular_bool
2168             { \skip_horizontal:n { - \tabcolsep } }
2169             { \skip_horizontal:n { - \arraycolsep } }
2170             \@@_use_arraybox_with_notes_c:
2171             \bool_if:NTF \l_@@_tabular_bool
2172             { \skip_horizontal:n { - \tabcolsep } }
2173             { \skip_horizontal:n { - \arraycolsep } }
2174         }

```

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

```

2175         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2176     }
2177     \exp_after:wN \right \g_@@_right_delim_tl
2178     $ % $
2179 }

```

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.

```

2180     \bool_if:NTF \l_@@_delimiters_max_width_bool
2181     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2182     \@@_put_box_in_flow:
2183 }

```

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

```

2184     \bool_if:NT \g_@@_last_col_found_bool
2185     { \skip_horizontal:N \g_@@_width_last_col_dim }
2186     \bool_if:NT \l_@@_preamble_bool
2187     {
2188         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2189         { \@@_err_columns_not_used: }
2190     }
2191     \@@_after_array:

```

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

```

2192     \egroup

```

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

```

2193     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2194     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2195     \iow_now:Ne \@mainaux
2196     {
2197         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2198         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2199         { \exp_not:o \g_@@_aux_tl }
2200     }
2201     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2202     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2203 }

```

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

```

2204 \cs_new_protected:Npn \@@_err_columns_not_used:
2205 {
2206     \@@_warning:n { columns~not~used }
2207     \cs_gset:Npn \@@_err_columns_not_used: { }
2208 }

```

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 .

```

2209 \cs_new_protected:Npn \@@_compute_width_X:
2210 {
2211   \tl_gput_right:Ne \g_@@_aux_tl
2212   {
2213     \bool_set_true:N \l_@@_X_columns_aux_bool
2214     \dim_set:Nn \l_@@_X_columns_dim
2215     {

```

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`

```

2216     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2217     { \dim_use:N \l_@@_X_columns_dim }
2218     {
2219       \dim_compare:nNnTF
2220       {
2221         \dim_abs:n
2222         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2223       }
2224       <
2225       { 0.001 pt }
2226       { \dim_use:N \l_@@_X_columns_dim }
2227       {
2228         \dim_eval:n
2229         {
2230           \l_@@_X_columns_dim
2231           +
2232           \fp_to_dim:n
2233           {
2234             (
2235               \dim_eval:n
2236               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2237             )
2238             / \fp_use:N \g_@@_total_X_weight_fp
2239           }
2240         }
2241       }
2242     }
2243   }
2244 }
2245 }

```

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`.

```

2246 \cs_new_protected:Npn \@@_transform_preamble:
2247 {
2248   \@@_transform_preamble_i:
2249   \@@_transform_preamble_ii:
2250 }

```

```

2251 \cs_new_protected:Npn \@@_transform_preamble_i:
2252 {
2253   \int_gzero:N \c@jCol

```

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

```

2254   \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.

```

2255   \bool_gset_false:N \g_tmpb_bool

```

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

```

2256   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2257   \int_zero:N \l_tmpa_int
2258   \tl_gclear:N \g_@@_array_preamble_tl
2259   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2260   {
2261     \tl_gset:Nn \g_@@_array_preamble_tl
2262       { ! { \skip_horizontal:N \arrayrulewidth } }
2263   }
2264   {
2265     \clist_if_in:NnT \l_@@_vlines_clist 1
2266     {
2267       \tl_gset:Nn \g_@@_array_preamble_tl
2268         { ! { \skip_horizontal:N \arrayrulewidth } }
2269     }
2270   }

```

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

```

2271   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2272   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2273   \@@_replace_columncolor:
2274 }

```

```

2275 \cs_new_protected:Npn \@@_transform_preamble_ii:
2276 {

```

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

```

2277   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2278   {
2279     \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2280     { \bool_gset_true:N \g_@@_delims_bool }
2281   }
2282   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2283   \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).

```

2284   \int_if_zero:nTF \l_@@_first_col_int
2285   { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2286   {
2287     \bool_if:NF \g_@@_delims_bool
2288     {
2289       \bool_if:NF \l_@@_tabular_bool
2290       {
2291         \clist_if_empty:NT \l_@@_vlines_clist

```

```

2292         {
2293             \bool_if:NF \l_@@_exterior_arraycolsep_bool
2294             { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2295         }
2296     }
2297 }
2298 }
2299 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2300 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2301 {
2302     \bool_if:NF \g_@@_delims_bool
2303     {
2304         \bool_if:NF \l_@@_tabular_bool
2305         {
2306             \clist_if_empty:NT \l_@@_vlines_clist
2307             {
2308                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2309                 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2310             }
2311         }
2312     }
2313 }

```

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.

```

2314     \tag_if_active:F
2315     {

```

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

```

2316     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2317     {
2318         \tl_gput_right:Nn \g_@@_array_preamble_tl
2319         { > { \@@_err_too_many_cols: } 1 }
2320     }
2321 }
2322 }

```

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.

```

2323 \cs_new_protected:Npn \@@_rec_preamble:n #1
2324 {

```

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

```

2325     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2326     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2327     {

```

Now, the columns defined by \newcolumntype of array.

```

2328     \cs_if_exist:cTF { NC @ find @ #1 }
2329     {
2330         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }

```

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.


```

2331         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2332     }
2333     {
2334         \str_if_eq:nnTF { #1 } { S }
2335         { \@@_fatal:n { unknown~column~type~S } }
2336         { \@@_fatal:nn { unknown~column~type } { #1 } }
2337     }
2338 }
2339 }

```

For c, l and r

```

2340 \cs_new_protected:Npn \@@_c: #1
2341 {
2342     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2343     \tl_gclear:N \g_@@_pre_cell_tl
2344     \tl_gput_right:Nn \g_@@_array_preamble_tl
2345     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2346     \int_gincr:N \c@jCol
2347     \@@_rec_preamble_after_col:n
2348 }

2349 \cs_new_protected:Npn \@@_l: #1
2350 {
2351     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2352     \tl_gclear:N \g_@@_pre_cell_tl
2353     \tl_gput_right:Nn \g_@@_array_preamble_tl
2354     {
2355         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2356         l
2357         < \@@_cell_end:
2358     }
2359     \int_gincr:N \c@jCol
2360     \@@_rec_preamble_after_col:n
2361 }

2362 \cs_new_protected:Npn \@@_r: #1
2363 {
2364     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2365     \tl_gclear:N \g_@@_pre_cell_tl
2366     \tl_gput_right:Nn \g_@@_array_preamble_tl
2367     {
2368         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2369         r
2370         < \@@_cell_end:
2371     }
2372     \int_gincr:N \c@jCol
2373     \@@_rec_preamble_after_col:n
2374 }

```

For ! and @

```

2375 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2376 {
2377     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2378     \@@_rec_preamble:n
2379 }
2380 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2381 \cs_new_protected:cpn { @@ _ | : } #1
2382 {

```

`\l_tmpa_int` is the number of successive occurrences of |

```

2383 \int_incr:N \l_tmpa_int
2384 \@@_make_preamble_i_i:n
2385 }
2386 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2387 {

```

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

```

2388 \str_if_eq:nnTF { #1 } { | }
2389 { \use:c { @@ _ | : } | }
2390 { \@@_make_preamble_i_ii:nn { } #1 }
2391 }

```

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

```

2392 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2393 {
2394 \str_if_eq:nnTF { #2 } { [ ] }
2395 { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2396 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2397 }
2398 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2399 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2400 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2401 {
2402 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2403 \tl_gput_right:Ne \g_@@_array_preamble_tl
2404 {

```

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

```

2405 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2406 }
2407 \tl_gput_right:Ne \g_@@_rules_tl
2408 {

```

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:

```

2409 {
2410 \int_compare:nNnT \l_tmpa_int > 1
2411 { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2412 \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2413 \dim_set:Nn \l_@@_rule_width_after_dim
2414 { \dim_use:N \l_@@_rule_width_before_dim }
2415 \@@_draw_vrule:n { #2 }
2416 }

```

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.

```

2417 }
2418 \int_zero:N \l_tmpa_int
2419 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2420 \@@_rec_preamble:n #1
2421 }

```

```

2422 \cs_new_protected:cpn { @@ _ > : } #1 #2
2423 {
2424   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2425   \@@_rec_preamble:n
2426 }
2427 \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.

```

2428 \keys_define:nn { nicematrix / p-column }
2429 {
2430   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2431   r .value_forbidden:n = true ,
2432   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2433   c .value_forbidden:n = true ,
2434   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2435   l .value_forbidden:n = true ,
2436   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2437   S .value_forbidden:n = true ,
2438   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2439   p .value_forbidden:n = true ,
2440   t .meta:n = p ,
2441   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2442   m .value_forbidden:n = true ,
2443   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2444   b .value_forbidden:n = true
2445 }

```

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

```

2446 \cs_new_protected:Npn \@@_p: #1
2447 {
2448   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2449   \@@_make_preamble_ii_i:n
2450 }
2451 \cs_new_eq:NN \@@_b: \@@_p:
2452 \cs_new_eq:NN \@@_m: \@@_p:
2453 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2454 {
2455   \str_if_eq:nnTF { #1 } { [ ]
2456     { \@@_make_preamble_ii_ii:w [ ] }
2457     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2458   }
2459 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2460 { \@@_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.

```

2461 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2462 {

```

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).

```

2463   \str_set:Nn \l_@@_hpos_col_str { j }
2464   \@@_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`.

```

2465   \setlength { \l_tmpa_dim } { #2 }

```

```

2466 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2467 }
2468 \cs_new_protected:Npn \@@_keys_p_column:n #1
2469 { \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.

```

2470 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2471 {

```

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

```

2472 \use:e
2473 {
2474 \@@_make_preamble_ii_vi:nnnnnnnn
2475 { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2476 { #1 }
2477 {
2478 \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.

```

2479 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2480 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2481 {

```

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

```

2482 \def \exp_not:N \l_@@_hpos_cell_tl
2483 { \str_lowercase:f { \l_@@_hpos_col_str } }
2484 }
2485 \IfPackageLoadedTF { ragged2e }
2486 {
2487 \str_case:on \l_@@_hpos_col_str
2488 {

```

The following `\exp_not:N` are mandatory.

```

2489 c { \exp_not:N \Centering }
2490 l { \exp_not:N \RaggedRight }
2491 r { \exp_not:N \RaggedLeft }
2492 }
2493 }
2494 {
2495 \str_case:on \l_@@_hpos_col_str
2496 {
2497 c { \exp_not:N \centering }
2498 l { \exp_not:N \raggedright }
2499 r { \exp_not:N \raggedleft }
2500 }
2501 }
2502 #3
2503 }
2504 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2505 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2506 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2507 { #2 }
2508 {
2509 \str_case:onF \l_@@_hpos_col_str
2510 {
2511 { j } { c }
2512 { si } { c }
2513 }

```

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

```

2514         { \str_lowercase:f \l_@@_hpos_col_str }
2515     }
2516 }

```

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

```

2517     \int_gincr:N \c@jCol
2518     \@@_rec_preamble_after_col:n
2519 }

```

#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*.

```

2520 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2521 {
2522     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2523     {
2524         \tl_gput_right:Nn \g_@@_array_preamble_tl
2525         { > \@@_test_if_empty_for_S: }
2526     }
2527     {
2528         \str_if_eq:eeTF { #7 } { varwidth }
2529         {
2530             \tl_gput_right:Nn \g_@@_array_preamble_tl
2531             { > \@@_test_if_empty_varwidth: }
2532         }
2533         { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2534     }
2535     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2536     \tl_gclear:N \g_@@_pre_cell_tl
2537     \tl_gput_right:Nn \g_@@_array_preamble_tl
2538     {
2539         > {

```

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.

```

2540         \dim_set:Nn \l_@@_col_width_dim { #2 }
2541         \@@_cell_begin:

```

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

```

2542         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2543     \everypar
2544     {
2545         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2546         \everypar { }
2547     }

```

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

```

2548     #3

```

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

```

2549         \g_@@_row_style_tl
2550         \arraybackslash
2551         #5
2552     }
2553     #8
2554     < {
2555         #6

```

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

```

2556         \@finalstrut \@arstrutbox
2557         \use:c { end #7 }

```

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

```

2558         #4
2559         \@@_cell_end:
2560     }
2561 }
2562 }

```

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

```

2563 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2564 {

```

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 `&`).

```

2565     \group_align_safe_begin:
2566     \peek_meaning:NTF &
2567     \@@_the_cell_is_empty:
2568     {
2569         \peek_meaning:NTF \\\
2570         \@@_the_cell_is_empty:
2571         {
2572             \peek_meaning:NTF \crcr
2573             \@@_the_cell_is_empty:
2574             \group_align_safe_end:
2575         }
2576     }
2577 }

```

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

```

2578 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2579 {
2580     \group_align_safe_begin:
2581     \peek_meaning:NTF &
2582     \@@_the_cell_is_empty_varwidth:
2583     {
2584         \peek_meaning:NTF \\\
2585         \@@_the_cell_is_empty_varwidth:
2586         {
2587             \peek_meaning:NTF \crcr
2588             \@@_the_cell_is_empty_varwidth:
2589             \group_align_safe_end:
2590         }
2591     }
2592 }

2593 \cs_new_protected:Npn \@@_the_cell_is_empty:
2594 {
2595     \group_align_safe_end:
2596     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2597     {

```

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

```
2598 \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).

```
2599 \skip_horizontal:N \l_@@_col_width_dim
2600 }
2601 }
2602 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2603 {
2604   \group_align_safe_end:
2605   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2606     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2607 }
2608 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2609 {
2610   \peek_meaning:NT \__siunitx_table_skip:n
2611   { \bool_gset_true:N \g_@@_empty_cell_bool }
2612 }
```

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

```
2613 \cs_new_protected:Npn \@@_center_cell_box:
2614 {
```

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

```
2615   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2616   {
2617     \dim_compare:nNnT
2618       { \box_ht:N \l_@@_cell_box }
2619     >
```

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).

```
2620     { \box_ht:N \strutbox }
2621     {
2622       \hbox_set:Nn \l_@@_cell_box
2623       {
2624         \box_move_down:nn
2625         {
2626           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2627             + \baselineskip ) / 2
2628         }
2629         { \box_use:N \l_@@_cell_box }
2630       }
2631     }
2632   }
2633 }
```

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

```
2634 \cs_new_protected:Npn \@@_V: #1 #2
2635 {
2636   \str_if_eq:nnTF { #2 } { [ ] }
2637   { \@@_make_preamble_V_i:w [ ] }
2638   { \@@_make_preamble_V_i:w [ ] { #2 } }
2639 }
```

```

2640 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2641 { \@@_make_preamble_V_ii:nn { #1 } }
2642 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2643 {
2644   \str_set:Nn \l_@@_vpos_col_str { p }
2645   \str_set:Nn \l_@@_hpos_col_str { j }
2646   \@@_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`.

```

2647   \setlength { \l_tmpa_dim } { #2 }
2648   \IfPackageLoadedTF { varwidth }
2649   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2650   {
2651     \@@_error_or_warning:n { varwidth-not-loaded }
2652     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2653   }
2654 }
2655 % \end{macrocod}
2656 %
2657 % \medskip
2658 % For |w| and |W|
2659 % \begin{macrocode}
2660 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2661 \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.

```

2662 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2663 {
2664   \tl_if_in:nnTF { clrs } { #3 }
2665   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2666   {
2667     \@@_error:nn { Invalid-argument-for-w } { #3 }
2668     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2669   }
2670 }
2671 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2672 {
2673   \str_if_eq:nnTF { #3 } { s }
2674   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2675   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2676 }

```

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.

```

2677 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2678 {
2679   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2680   \tl_gclear:N \g_@@_pre_cell_tl
2681   \tl_gput_right:Nn \g_@@_array_preamble_tl
2682   {
2683     > {

```

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`.


```

2684         \setlength { \l_@@_col_width_dim } { #2 }
2685         \@@_cell_begin:
2686         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2687     }
2688     c
2689     < {
2690         \@@_cell_end_for_w_s:
2691         #1
2692         \@@_adjust_size_box:
2693         \box_use_drop:N \l_@@_cell_box
2694     }
2695 }
2696 \int_gincr:N \c@jCol
2697 \@@_rec_preamble_after_col:n
2698 }

```

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

```

2699 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2700 {
2701     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2702     \tl_gclear:N \g_@@_pre_cell_tl
2703     \tl_gput_right:Nn \g_@@_array_preamble_tl
2704     {
2705         > {

```

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`.

```

2706         \setlength { \l_@@_col_width_dim } { #4 }
2707         \hbox_set:Nw \l_@@_cell_box
2708         \@@_cell_begin:
2709         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2710     }
2711     c
2712     < {
2713         \@@_cell_end:
2714         \hbox_set_end:
2715         #1
2716         \@@_adjust_size_box:
2717         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2718     }
2719 }

```

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

```

2720     \int_gincr:N \c@jCol
2721     \@@_rec_preamble_after_col:n
2722 }

```

```

2723 \cs_new_protected:Npn \@@_special_W:
2724 {
2725     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2726     { \@@_warning:n { W~warning } }
2727 }

```

For S (of siunitx).

```

2728 \AtBeginDocument
2729 {
2730     \IfPackageLoadedT { siunitx }
2731     {
2732         \cs_new_protected:Npn \@@_S: #1 #2

```

```

2733         {
2734             \str_if_eq:nnTF { #2 } { [ ] }
2735             { \@@_make_preamble_S:w [ ] }
2736             { \@@_make_preamble_S:w [ ] { #2 } }
2737         }
2738     }
2739 }

2740 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2741 { \@@_make_preamble_S_i:n { #1 } }

2742 \cs_new_protected:Npn \@@_test_siunitx:
2743 {
2744     \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2745     { \@@_fatal:n { siunitx~too~old } }
2746     \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2747 }
2748 % \end{macrocode}
2749 %
2750 % \begin{macrocode}
2751 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2752 {
2753     \@@_test_siunitx:
2754     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2755     \tl_gclear:N \g_@@_pre_cell_tl
2756     \tl_gput_right:Nn \g_@@_array_preamble_tl
2757     {
2758         > {

```

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).

```

2759         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2760         \keys_set:nn { siunitx } { #1 }
2761         \@@_cell_begin:
2762         \siunitx_cell_begin:w
2763     }
2764     c
2765     <
2766     {
2767         \siunitx_cell_end:

```

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

```

2768         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2769         {
2770             \bool_if:NTF \l__siunitx_table_text_bool
2771             \bool_set_true:N
2772             \bool_set_false:N
2773             \l__siunitx_table_text_bool
2774         }
2775         \@@_cell_end:
2776     }
2777 }

```

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

```

2778     \int_gincr:N \c@jCol
2779     \@@_rec_preamble_after_col:n
2780 }

```

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

```

2781 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2

```

```

2782 {
2783   \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.

```

2784   \int_if_zero:nTF \c@jCol
2785   {
2786     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2787     {

```

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

```

2788       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2789       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2790       \@@_rec_preamble:n #2
2791     }
2792   {
2793     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2794     \@@_make_preamble_iv:nn { #1 } { #2 }
2795   }
2796 }
2797 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2798 }
2799 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2800 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2801 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2802 {
2803   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2804   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2805   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2806   {
2807     \@@_error:nn { delimiter~after~opening } { #2 }
2808     \@@_rec_preamble:n
2809   }
2810   { \@@_rec_preamble:n #2 }
2811 }

```

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

```

2812 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2813 { \use:c { @@ _ \token_to_str:N ( : } }

```

For the closing delimiters. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have an opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is {NiceArray}).

```

2814 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2815 {
2816   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2817   \tl_if_in:nnTF { ) ] \} } { #2 }
2818   { \@@_make_preamble_v:nnn #1 #2 }
2819   {
2820     \str_if_eq:nnTF \s_stop { #2 }
2821     {
2822       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2823       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2824       {
2825         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2826         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2827         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2828         \@@_rec_preamble:n #2
2829       }
2830     }
2831   {
2832     \tl_if_in:nnT { ( [ \{ \left } { #2 }

```

```

2833         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2834     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2835     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2836     \@@_rec_preamble:n #2
2837 }
2838 }
2839 }
2840 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2841 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2842 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2843 {
2844     \str_if_eq:nnTF \s_stop { #3 }
2845     {
2846         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2847         {
2848             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2849             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2850             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2851             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2852         }
2853         {
2854             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2855             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2856             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2857             \@@_error:nn { double~closing~delimiter } { #2 }
2858         }
2859     }
2860     {
2861         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2862         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2863         \@@_error:nn { double~closing~delimiter } { #2 }
2864         \@@_rec_preamble:n #3
2865     }
2866 }
2867 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2868 { \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 @{...}.

```

2869 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2870 {
2871     \str_if_eq:nnTF { #1 } { < }
2872     { \@@_rec_preamble_after_col_i:n }
2873     {
2874         \str_if_eq:nnTF { #1 } { @ }
2875         { \@@_rec_preamble_after_col_ii:n }
2876         {
2877             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2878             {
2879                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2880                 { ! { \skip_horizontal:N \arrayrulewidth } }
2881             }
2882             {
2883                 \clist_if_in:NeT \l_@@_vlines_clist
2884                 { \int_eval:n { \c@jCol + 1 } }
2885                 {
2886                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2887                     { ! { \skip_horizontal:N \arrayrulewidth } }
2888                 }
2889             }
2890         }
2891     }
2892 }

```

```

2890         \@@_rec_preamble:n { #1 }
2891     }
2892 }
2893 }
2894 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2895 {
2896     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2897     \@@_rec_preamble_after_col:n
2898 }

```

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.

```

2899 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2900 {
2901     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2902     {
2903         \tl_gput_right:Nn \g_@@_array_preamble_tl
2904         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2905     }
2906     {
2907         \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2908         {
2909             \tl_gput_right:Nn \g_@@_array_preamble_tl
2910             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2911         }
2912         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2913     }
2914     \@@_rec_preamble:n
2915 }
2916 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2917 {
2918     \tl_clear:N \l_tmpa_tl
2919     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2920     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2921 }

```

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

```

2922 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2923 { \@@_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.

```

2924 \cs_new_protected:Npn \@@_X: #1 #2
2925 {
2926     \str_if_eq:nnTF { #2 } { [ ]
2927     { \@@_make_preamble_X:w [ ]
2928     { \@@_make_preamble_X:w [ ] #2 }
2929 }
2930 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2931 { \@@_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.

```

2932 \keys_define:nn { nicematrix / X-column }
2933 {

```

```

2934 V .code:n =
2935   \IfPackageLoadedTF { varwidth }
2936   {
2937     \bool_set_true:N \l_@@_V_of_X_bool
2938     \bool_gset_true:N \g_@@_V_of_X_bool
2939   }
2940   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2941   unknown .code:n =
2942     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2943     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2944     { \@@_error_or_warning:n { invalid~weight } }
2945 }

```

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

```

2946 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2947 {

```

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).

```

2948   \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).

```

2949   \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`).

```

2950   \fp_set:Nn \l_tmpa_fp { 1.0 }
2951   \@@_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.

```

2952   \bool_set_false:N \l_@@_V_of_X_bool
2953   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2954   \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).

```

2955   \bool_if:NTF \l_@@_X_columns_aux_bool
2956   {
2957     \@@_make_preamble_ii_iv:nnn

```

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

```

2958     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2959     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2960     { \@@_no_update_width: }
2961   }

```

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.

```

2962   {
2963     \tl_gput_right:Nn \g_@@_array_preamble_tl
2964     {
2965       > {
2966         \@@_cell_begin:
2967         \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`.

```

2968   \NotEmpty

```

The following code will nullify the box of the cell.

```
2969          \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2970          { \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`.

```
2971          \begin { minipage } { 5 cm } \arraybackslash
2972          }
2973          c
2974          < {
2975              \end { minipage }
2976              \@@_cell_end:
2977          }
2978      }
2979      \int_gincr:N \c@jCol
2980      \@@_rec_preamble_after_col:n
2981  }
2982  }

2983 \cs_new_protected:Npn \@@_no_update_width:
2984 {
2985     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2986     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2987 }
```

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

```
2988 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2989 {
2990     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2991     { \int_eval:n { \c@jCol + 1 } }
2992     \tl_gput_right:Ne \g_@@_array_preamble_tl
2993     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2994     \@@_rec_preamble:n
2995 }
```

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.

```
2996 \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).

```
2997 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2998 { \@@_fatal:n { Preamble-forgotten } }
2999 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3000 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3001 { @@ _ \token_to_str:N \hline : }
3002 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3003 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3004 { @@ _ \token_to_str:N \hline : }
3005 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3006 { @@ _ \token_to_str:N \hline : }
3007 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3008 { @@ _ \token_to_str:N \hline : }
3009 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3010 { @@ _ \token_to_str:N \hline : }
3011 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3012 { \@@_fatal:n { NiceTabularX-probably-required } }
3013 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3014 { @@ _ \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).

```
3015 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3016 {
```

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`.

```
3017 \multispan { #1 }
3018 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3019 \begingroup
3020 \tbl_update_multicolumn_cell_data:n { #1 }
```

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

```
3021 \tl_gclear:N \g_@@_preamble_tl
3022 \@@_make_m_preamble:n #2 \q_stop
```

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

```
3023 \def \@addamp
3024 {
3025   \legacy_if:nTF { @firstamp }
3026   { \legacy_if_set_false:n { @firstamp } }
3027   { \@preamerr 5 }
3028 }
3029 \exp_args:No \@mkpream \g_@@_preamble_tl
3030 \@addtopreamble \@empty
3031 \endgroup
3032 \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
3033 \int_compare:nNnT { #1 } > 1
3034 {
3035   \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3036   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3037   \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3038   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3039   {
3040     {
3041       \int_if_zero:nTF \c@jCol
3042       { \int_eval:n { \c@iRow + 1 } }
3043       { \int_use:N \c@iRow }
3044     }
3045     { \int_eval:n { \c@jCol + 1 } }
3046     {
3047       \int_if_zero:nTF \c@jCol
3048       { \int_eval:n { \c@iRow + 1 } }
3049       { \int_use:N \c@iRow }
3050     }
3051     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```
3052 { }
3053 }
3054 }
```

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

```
3055 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3056 {
```



```

3057 \tl_gput_right:N\g_@@_pre_code_before_tl
3058 {
3059 \@@_rectanglecolor [ ##1 ]
3060 { \exp_not:n { ##2 } }
3061 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3062 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3063 }
3064 \ignorespaces
3065 }

```

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

```

3066 \def \@sharp { #3 }
3067 \@arstrut
3068 \@preamble
3069 \null

```

We add some lines.

```

3070 \int_gadd:Nn \c@jCol { #1 - 1 }
3071 \int_compare:nNt \c@jCol > \g_@@_col_total_int
3072 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3073 \ignorespaces
3074 }

```

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`.

```

3075 \
3076 \cs_new_protected:Npn \@_make_m_preamble:n #1
3077 {
3078 \str_case:nnF { #1 }
3079 {
3080 c { \@_make_m_preamble_i:n #1 }
3081 l { \@_make_m_preamble_i:n #1 }
3082 X { \@_make_m_preamble_i:n l }
3083 r { \@_make_m_preamble_i:n #1 }
3084 > { \@_make_m_preamble_ii:nn #1 }
3085 ! { \@_make_m_preamble_ii:nn #1 }
3086 @ { \@_make_m_preamble_ii:nn #1 }
3087 | { \@_make_m_preamble_iii:n #1 }
3088 p { \@_make_m_preamble_iv:nnn t #1 }
3089 m { \@_make_m_preamble_iv:nnn c #1 }
3090 b { \@_make_m_preamble_iv:nnn b #1 }
3091 w { \@_make_m_preamble_v:nnnn { } #1 }
3092 W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
3093 \q_stop { }
3094 }
3095 {
3096 \cs_if_exist:cTF { NC @ find @ #1 }
3097 {
3098 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3099 \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
3100 }
3101 {
3102 \str_if_eq:nnTF { #1 } { S }
3103 { \@_fatal:n { unknown~column~type~S~multicolumn } }
3104 { \@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3105 }
3106 }
3107 }

```

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

```

3108 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
3109 {

```

```

3110 \tl_gput_right:Nn \g_@@_preamble_tl
3111 {
3112   > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3113   #1
3114   < \@@_cell_end:
3115 }

```

We test for the presence of a <.

```

3116 \@@_make_m_preamble_x:n
3117 }

```

For >, ! and @

```

3118 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3119 {
3120   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3121   \@@_make_m_preamble:n
3122 }

```

For |

```

3123 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3124 {
3125   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3126   \@@_make_m_preamble:n
3127 }

```

For p, m and b

```

3128 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3129 {
3130   \tl_gput_right:Nn \g_@@_preamble_tl
3131   {
3132     > {
3133       \@@_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`.

```

3134 \setlength { \l_tmpa_dim } { #3 }
3135 \begin { minipage } [ #1 ] { \l_tmpa_dim }
3136 \mode_leave_vertical:
3137 \arraybackslash
3138 \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3139 }
3140 c
3141 < {
3142   \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3143   \end { minipage }
3144   \@@_cell_end:
3145 }
3146 }

```

We test for the presence of a <.

```

3147 \@@_make_m_preamble_x:n
3148 }

```

For w and W

```

3149 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3150 {
3151   \tl_gput_right:Nn \g_@@_preamble_tl
3152   {
3153     > {
3154       \dim_set:Nn \l_@@_col_width_dim { #4 }
3155       \hbox_set:Nw \l_@@_cell_box
3156       \@@_cell_begin:
3157       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }

```

```

3158     }
3159     c
3160     < {
3161         \@@_cell_end:
3162         \hbox_set_end:
3163         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3164         #1
3165         \@@_adjust_size_box:
3166         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3167     }
3168 }

```

We test for the presence of a <.

```

3169     \@@_make_m_preamble_x:n
3170 }

```

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

```

3171 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3172 {
3173     \str_if_eq:nnTF { #1 } { < }
3174     \@@_make_m_preamble_ix:n
3175     { \@@_make_m_preamble:n { #1 } }
3176 }
3177 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3178 {
3179     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3180     \@@_make_m_preamble_x:n
3181 }

```

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).

```

3182 \cs_new_protected:Npn \@@_put_box_in_flow:
3183 {
3184     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3185     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3186     \str_if_eq:eeTF \l_@@_baseline_tl { c }
3187     { \box_use_drop:N \l_tmpa_box }
3188     \@@_put_box_in_flow_i:
3189 }

```

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

```

3190 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3191 {
3192     \pgfpicture
3193     \@@_qpoint:n { row - 1 }
3194     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3195     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3196     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3197     \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).

```

3198     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3199     {
3200         \int_set:Nn \l_tmpa_int
3201         { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3202         \bool_lazy_or:nnT
3203         { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3204         { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }

```

```

3205         {
3206             \@@_error:n { bad-value-for-baseline-line }
3207             \int_set:Nn \l_tmpa_int 1
3208         }
3209     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3210 }
3211 {
3212     \str_if_eq:eeTF t \l_@@_baseline_tl
3213     { \int_set:Nn \l_tmpa_int 1 }
3214     {
3215         \str_if_eq:eeTF b \l_@@_baseline_tl
3216         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3217         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3218     }
3219     \bool_lazy_or:nnT
3220     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3221     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3222     {
3223         \@@_error:n { bad-value-for-baseline }
3224         \int_set:Nn \l_tmpa_int 1
3225     }
3226     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3227         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3228     }
3229     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3230     \endpgfpicture
3231     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3232     % \box_use_drop:N \l_tmpa_box % 2026/04/13
3233 }

```

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).

```

3234 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3235 {

```

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.

```

3236     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3237     {
3238         \int_compare:nNnT \c@jCol > 1
3239         {
3240             \box_set_wd:Nn \l_@@_the_array_box
3241             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3242         }
3243     }

```

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

```

3244     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3245     \bool_if:NT \l_@@_caption_above_bool
3246     {
3247         \tl_if_empty:NF \l_@@_caption_tl
3248         {
3249             \bool_set_false:N \g_@@_caption_finished_bool
3250             \int_gzero:N \c@tabularnote
3251             \@@_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).

```

3252         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3253         {
3254             \tl_gput_right:Ne \g_@@_aux_tl
3255             {
3256                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3257                 { \int_use:N \g_@@_notes_caption_int }
3258             }
3259             \int_gzero:N \g_@@_notes_caption_int
3260         }
3261     }
3262 }

```

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

```

3263     \hbox
3264     {
3265         \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.

```

3266         \@@_create_extra_nodes:
3267         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3268     }

```

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).

```

3269     \bool_lazy_any:nT
3270     {
3271         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3272         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3273         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3274     }
3275     {
3276         \bool_if:NTF \l_@@_notes_no_print_bool
3277         { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3278         \@@_tabular_notes:
3279     }
3280     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3281     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3282     \end { minipage }
3283 }

```

```

3284 \cs_new_protected:Npn \@@_insert_caption:
3285 {
3286     \tl_if_empty:NF \l_@@_caption_tl
3287     {
3288         \cs_if_exist:NTF \@capttype
3289         { \@@_insert_caption_i: }
3290         { \@@_error:n { caption~outside~float } }
3291     }
3292 }

```

```

3293 \cs_new_protected:Npn \@@_insert_caption_i:
3294 {
3295     \group_begin:

```

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

```
3296 \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.

```
3297 \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3298 \tl_if_empty:NTF \l_@@_short_caption_tl
3299 \caption
3300 { \caption [ \l_@@_short_caption_tl ] }
3301 { \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.

```
3302 \bool_if:NF \g_@@_caption_finished_bool
3303 {
3304   \bool_gset_true:N \g_@@_caption_finished_bool
3305   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3306   \int_gzero:N \c@tabularnote
3307 }
3308 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3309 \group_end:
3310 }

3311 \cs_new_protected:Npn \@_tabularnote_error:n #1
3312 {
3313   \@_error_or_warning:n { tabularnote-below-the-tabular }
3314   \cs_gset:Npn \@_tabularnote_error:n ##1 { }
3315 }

3316 \cs_set_protected:Npn \@_tabular_notes_error:
3317 { \@_error:n { Bad-use-of-NiceTabularNotes } }

3318 \cs_set_eq:NN \NiceTabularNotes \@_tabular_notes_error:

3319 \cs_set_protected:Npn \@_tabular_notes:
3320 {
3321   \cs_gset_eq:NN \NiceTabularNotes \@_tabular_notes_error:
3322   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3323   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3324   \skip_vertical:N 0.65ex
```

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

```
3325 \group_begin:
3326 \l_@@_notes_code_before_tl
3327 \tl_if_empty:NF \g_@@_tabularnote_tl
3328 {
3329   \g_@@_tabularnote_tl \par
3330   \tl_gclear:N \g_@@_tabularnote_tl
3331 }
```

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.

```
3332 \int_compare:nNnT \c@tabularnote > \c_zero_int
3333 {
3334   \bool_if:NTF \l_@@_notes_para_bool
3335   {
3336     \begin { tabularnotes* }
3337     \seq_map_inline:Nn \g_@@_notes_seq
3338       { \@_one_tabularnote:nn ##1 }
3339     \strut
3340     \end { tabularnotes* }
```

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

```

3341         \par
3342     }
3343     {
3344         \tabularnotes
3345         \seq_map_inline:Nn \g_@@_notes_seq
3346         { \@@_one_tabularnote:nn ##1 }
3347         \strut
3348         \endtabularnotes
3349     }
3350 }
3351 \unskip
3352 \group_end:
3353 \bool_if:NT \l_@@_notes_bottomrule_bool
3354 {
3355     \IfPackageLoadedTF { booktabs }
3356     {

```

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

```

3357         \skip_vertical:N \aboverulesep

```

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

```

3358         { \CT@arc@ \hrule height \heavyrulewidth }
3359     }
3360     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3361 }
3362 \l_@@_notes_code_after_tl
3363 \seq_gclear:N \g_@@_notes_seq
3364 \seq_gclear:N \g_@@_notes_in_caption_seq
3365 \int_gzero:N \c@tabularnote
3366 }

```

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.

```

3367 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3368 {
3369     \tl_if_novalue:nTF { #1 }
3370     { \item }
3371     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3372 }

```

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`.

```

3373 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3374 {
3375     \pgfpicture
3376     \@@_qpoint:n { row - 1 }
3377     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3378     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3379     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3380     \endpgfpicture
3381     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3382     \int_if_zero:nT \l_@@_first_row_int
3383     {
3384         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3385         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3386     }
3387     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3388 }

```

Now, the general case.

```
3389 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3390 {
```

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

```
3391 \str_if_eq:eeT \l_@@_baseline_tl { t }
3392 { \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`.

```
3393 \pgfpicture
3394 \@@_qpoint:n { row - 1 }
3395 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3396 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3397 {
3398   \int_set:Nn \l_tmpa_int
3399   {
3400     \str_range:Nnn
3401     \l_@@_baseline_tl
3402     { 6 }
3403     { \tl_count:o \l_@@_baseline_tl }
3404   }
3405   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3406 }
3407 {
3408   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3409   \bool_lazy_or:nnT
3410   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3411   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3412   {
3413     \@@_error:n { bad~value~for~baseline }
3414     \int_set:Nn \l_tmpa_int 1
3415   }
3416   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3417 }
3418 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3419 \endpgfpicture
3420 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3421 \int_if_zero:nT \l_@@_first_row_int
3422 {
3423   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3424   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3425 }
3426 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3427 }
```

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.

```
3428 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3429 {
```

We will compute the real width of both delimiters used.

```
3430 \dim_zero_new:N \l_@@_real_left_delim_dim
3431 \dim_zero_new:N \l_@@_real_right_delim_dim
3432 \hbox_set:Nn \l_tmpb_box
3433 {
3434   \m@th
3435   $ % $
3436   \left #1
3437   \vcenter
3438   {
3439     \vbox_to_ht:nn
3440     { \box_ht_plus_dp:N \l_tmpa_box }
```



```

3441         { }
3442     }
3443     \right .
3444     $ % $
3445 }
3446 \dim_set:Nn \l_@@_real_left_delim_dim
3447 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3448 \hbox_set:Nn \l_tmpb_box
3449 {
3450     \m@th
3451     $ % $
3452     \left .
3453     \vbox_to_ht:nn
3454     { \box_ht_plus_dp:N \l_tmpa_box }
3455     { }
3456     \right #2
3457     $ % $
3458 }
3459 \dim_set:Nn \l_@@_real_right_delim_dim
3460 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3461 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3462 \@@_put_box_in_flow:
3463 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3464 }

```

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).

```

3465 \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).

```

3466 {
3467     \peek_remove_spaces:n
3468     {
3469         \peek_meaning:NTF \end
3470         \@@_analyze_end:Nn
3471         {
3472             \@@_transform_preamble:
3473             \@@_array:o \g_@@_array_preamble_tl
3474         }
3475     }
3476 }
3477 {
3478     \@@_create_col_nodes:
3479     \endarray
3480 }

```

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

```

3481 \NewDocumentEnvironment { @@-light-syntax } { b }
3482 {

```

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`.

```

3483 \tl_if_empty:nT { #1 }

```

```

3484     { \@@_fatal:n { empty-environment } }
3485 \tl_if_in:nnT { #1 } { & }
3486     { \@@_fatal:n { ampersand-in~light-syntax } }
3487 \tl_if_in:nnT { #1 } { \ }
3488     { \@@_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`.

```

3489 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3490 }

```

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.

```

3491 {
3492   \@@_create_col_nodes:
3493   \endarray
3494 }
3495 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3496 {
3497   \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).

```

3498 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3499 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3500 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3501   \seq_set_split:Nee
3502   \seq_set_split:Non
3503   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3504 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3505 \tl_if_empty:NF \l_tmpa_tl
3506   { \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.

```

3507 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3508   { \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`.

```

3509 \tl_build_begin:N \l_@@_new_body_tl
3510 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3511 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3512 \@@_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).

```

3513 \seq_map_inline:Nn \l_@@_rows_seq
3514 {
3515   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3516   \@@_line_with_light_syntax:n { ##1 }
3517 }
3518 \tl_build_end:N \l_@@_new_body_tl

```

```

3519 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3520 {
3521     \int_set:Nn \l_@@_last_col_int
3522     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3523 }

```

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.

```

3524 \@@_transform_preamble:

3525 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3526 }

3527 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3528 {
3529     \seq_clear_new:N \l_@@_cells_seq
3530     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3531     \int_set:Nn \l_@@_nb_cols_int
3532     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3533     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3534     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3535     \seq_map_inline:Nn \l_@@_cells_seq
3536     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3537 }
3538 \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`.

```

3539 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3540 {
3541     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3542     { \@@_fatal:n { empty~environment } }

```

We reup in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```

3543 \end { #2 }
3544 }

```

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`).

```

3545 \cs_new:Npn \@@_create_col_nodes:
3546 {
3547     \crrc
3548     \int_if_zero:nT \l_@@_first_col_int
3549     {
3550         \omit
3551         \hbox_overlap_left:n
3552         {
3553             \bool_if:NT \l_@@_code_before_bool
3554             { \pgfsys@markposition { \@@_env: - col - 0 } }
3555             \pgfpicture
3556             \pgfrememberpicturepositiononpagetrue
3557             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3558             \str_if_empty:NF \l_@@_name_str
3559             { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3560             \endpgfpicture
3561             \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3562         }
3563         &
3564     }
3565     \omit

```

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

```
3566 \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`).

```
3567 \int_if_zero:nTF \l_@@_first_col_int
3568 {
3569   \@@_mark_position:n { 1 }
3570   \pgfpicture
3571   \pgfrememberpicturepositiononpagetrue
3572   \pgfcoordinate { \@@_env: - col - 1 }
3573   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3574   \str_if_empty:NF \l_@@_name_str
3575   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3576   \endpgfpicture
3577 }
3578 {
3579   \bool_if:NT \l_@@_code_before_bool
3580   {
3581     \hbox
3582     {
3583       \skip_horizontal:n { 0.5 \arrayrulewidth }
3584       \pgfsys@markposition { \@@_env: - col - 1 }
3585       \skip_horizontal:n { -0.5 \arrayrulewidth }
3586     }
3587   }
3588   \pgfpicture
3589   \pgfrememberpicturepositiononpagetrue
3590   \pgfcoordinate { \@@_env: - col - 1 }
3591   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3592   \@@_node_alias:n { 1 }
3593   \endpgfpicture
3594 }
```

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.

```
3595 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3596 \bool_if:NF \l_@@_auto_columns_width_bool
3597 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3598 {
3599   \bool_lazy_and:nnTF
3600   \l_@@_auto_columns_width_bool
3601   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3602   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3603   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3604   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3605 }
3606 \skip_horizontal:N \g_tmpa_skip
3607 \hbox
3608 {
3609   \@@_mark_position:n { 2 }
3610   \pgfpicture
3611   \pgfrememberpicturepositiononpagetrue
3612   \pgfcoordinate { \@@_env: - col - 2 }
3613   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3614   \@@_node_alias:n { 2 }
3615   \endpgfpicture
3616 }
```

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.

```

3617 \int_gset:Nn \g_tmpa_int 1
3618 \bool_if:NTF \g_@@_last_col_found_bool
3619 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3620 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3621 {
3622   &
3623   \omit
3624   \int_gincr:N \g_tmpa_int

```

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

```

3625 \skip_horizontal:N \g_tmpa_skip
3626 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3627 \pgfpicture
3628 \pgfrememberpicturepositiononpagetrue
3629 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3630 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3631 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3632 \endpgfpicture
3633 }

```

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).

```

3634 \bool_lazy_or:nnF
3635 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3636 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3637 {
3638   &
3639   \omit
3640   \skip_horizontal:N \g_tmpa_skip
3641   \int_gincr:N \g_tmpa_int
3642   \bool_lazy_any:nF
3643   {
3644     \g_@@_delims_bool
3645     \l_@@_tabular_bool
3646     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3647     \l_@@_exterior_arraycolsep_bool
3648     \l_@@_bar_at_end_of_pream_bool
3649   }
3650   { \skip_horizontal:n { - \col@sep } }
3651   \bool_if:NT \l_@@_code_before_bool
3652   {
3653     \hbox
3654     {
3655       \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.

```

3656 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3657 { \skip_horizontal:n { - \arraycolsep } }
3658 \pgfsys@markposition
3659 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3660 \skip_horizontal:n { 0.5 \arrayrulewidth }
3661 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3662 { \skip_horizontal:N \arraycolsep }
3663 }
3664 }
3665 \pgfpicture
3666 \pgfrememberpicturepositiononpagetrue
3667 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3668 {

```

```

3669         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3670         {
3671             \pgfpoint
3672             { - 0.5 \arrayrulewidth - \arraycolsep }
3673             \c_zero_dim
3674         }
3675         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3676     }
3677     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3678 \endpgfpicture
3679 }

3680 \bool_if:NT \g_@@_last_col_found_bool
3681 {
3682     \hbox_overlap_right:n
3683     {
3684         \skip_horizontal:N \g_@@_width_last_col_dim
3685         \skip_horizontal:N \col@sep
3686         \bool_if:NT \l_@@_code_before_bool
3687         {
3688             \pgfsys@markposition
3689             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3690         }
3691         \pgfpicture
3692         \pgfrememberpicturepositiononpagetrue
3693         \pgfcoordinate
3694         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3695         \pgfpintorigin
3696         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3697         \endpgfpicture
3698     }
3699 }
3700 }

3701 \cs_new_protected:Npn \@@_mark_position:n #1
3702 {
3703     \bool_if:NT \l_@@_code_before_bool
3704     {
3705         \hbox
3706         {
3707             \skip_horizontal:n { -0.5 \arrayrulewidth }
3708             \pgfsys@markposition { \@@_env: - col - #1 }
3709             \skip_horizontal:n { 0.5 \arrayrulewidth }
3710         }
3711     }
3712 }

3713 \cs_new_protected:Npn \@@_node_alias:n #1
3714 {
3715     \str_if_empty:NF \l_@@_name_str
3716     { \pgfnodelias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3717 }

```

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

```

3718 \tl_const:Nn \c_@@_preamble_first_col_tl
3719 {
3720     >
3721     {

```

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

```

3722 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3723 \bool_gset_true:N \g_@@_after_col_zero_bool
3724 \@@_begin_of_row:
3725 \hbox_set:Nw \l_@@_cell_box
3726 \@@_math_toggle:
3727 \@@_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”.

```

3728 \int_compare:nNnT \c@iRow > \c_zero_int
3729 {
3730   \bool_lazy_or:nnT
3731     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3732     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3733   {
3734     \l_@@_code_for_first_col_tl
3735     \xglobal \colorlet { nicematrix-first-col } { . }
3736   }
3737 }
3738 }

```

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`.

```

3739 1
3740 <
3741 {
3742   \@@_math_toggle:
3743   \hbox_set_end:
3744   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3745   \@@_adjust_size_box:
3746   \@@_update_for_first_and_last_row:

```

We actualise the width of the “first column” because we will use this width after the construction of the array.

```

3747 \dim_gset:Nn \g_@@_width_first_col_dim
3748 { \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.

```

3749 \hbox_overlap_left:n
3750 {
3751   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3752     \@@_node_cell:
3753     { \box_use_drop:N \l_@@_cell_box }
3754     \skip_horizontal:N \l_@@_left_delim_dim
3755     \skip_horizontal:N \l_@@_left_margin_dim
3756     \skip_horizontal:N \l_@@_extra_left_margin_dim
3757   }
3758   \bool_gset_false:N \g_@@_empty_cell_bool
3759   \skip_horizontal:n { -2 \col@sep }
3760 }
3761 }

```

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

```

3762 \tl_const:Nn \c_@@_preamble_last_col_tl
3763 {
3764   >
3765   {
3766     \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).

```

3767 \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.

```

3768     \bool_gset_true:N \g_@@_last_col_found_bool
3769     \int_gincr:N \c@jCol
3770     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3771     \hbox_set:Nw \l_@@_cell_box
3772     \@@_math_toggle:
3773     \@@_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”.

```

3774     \int_compare:nNnT \c@iRow > \c_zero_int
3775     {
3776       \bool_lazy_or:nnT
3777       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3778       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3779       {
3780         \l_@@_code_for_last_col_tl
3781         \xglobal \colorlet { nicematrix-last-col } { . }
3782       }
3783     }
3784   }
3785   1
3786   <
3787   {
3788     \@@_math_toggle:
3789     \hbox_set_end:
3790     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3791     \@@_adjust_size_box:
3792     \@@_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.

```

3793     \dim_gset:Nn \g_@@_width_last_col_dim
3794     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3795     \skip_horizontal:n { -2 \col@sep }

```

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

```

3796     \hbox_overlap_right:n
3797     {
3798       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3799       {
3800         \skip_horizontal:N \l_@@_right_delim_dim
3801         \skip_horizontal:N \l_@@_right_margin_dim
3802         \skip_horizontal:N \l_@@_extra_right_margin_dim
3803         \@@_node_cell:
3804       }
3805     }
3806     \bool_gset_false:N \g_@@_empty_cell_bool
3807   }
3808 }

```

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

```

3809 \NewDocumentEnvironment { NiceArray } { }
3810 {
3811   \bool_gset_false:N \g_@@_delims_bool
3812   \str_if_empty:NT \g_@@_name_env_str
3813   { \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).

```

3814   \NiceArrayWithDelims . .
3815 }
3816 { \endNiceArrayWithDelims }

```


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

```

3817 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3818 {
3819   \NewDocumentEnvironment { #1 NiceArray } { }
3820   {
3821     \bool_gset_true:N \g_@@_delims_bool
3822     \str_if_empty:NT \g_@@_name_env_str
3823     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3824     \@@_test_if_math_mode:
3825     \NiceArrayWithDelims #2 #3
3826   }
3827   { \endNiceArrayWithDelims }
3828 }
3829 \@@_def_env:NNN p ( )
3830 \@@_def_env:NNN b [ ]
3831 \@@_def_env:NNN B \{ \}
3832 \@@_def_env:NNN v \vert \vert
3833 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

13.1 Definition of `{pNiceMatrix}`

```

3834 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3835 {
3836   \bool_set_false:N \l_@@_preamble_bool
3837   \tl_clear:N \l_tmpa_tl
3838   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3839   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3840   \tl_put_right:Nn \l_tmpa_tl
3841   {
3842     *
3843     {
3844       \int_case:nnF \l_@@_last_col_int
3845       {
3846         { -2 } { \c@MaxMatrixCols }
3847         { -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).

```

3848       }
3849       { \int_eval:n { \l_@@_last_col_int - 1 } }
3850     }
3851     { #2 }
3852   }
3853   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3854   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3855 }
3856 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3857 \clist_map_inline:nn { p , b , B , v , V }
3858 {
3859   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3860   {
3861     \bool_gset_true:N \g_@@_delims_bool
3862     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3863     \int_if_zero:nT \l_@@_last_col_int
3864     {
3865       \bool_set_true:N \l_@@_last_col_without_value_bool
3866       \int_set:Nn \l_@@_last_col_int { -1 }

```

```

3867     }
3868     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3869     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3870 }
3871 { \use:c { end #1 NiceArray } }
3872 }

```

We define also an environment {NiceMatrix}

```

3873 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3874 {
3875     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3876     \int_if_zero:nT \l_@@_last_col_int
3877     {
3878         \bool_set_true:N \l_@@_last_col_without_value_bool
3879         \int_set:Nn \l_@@_last_col_int { -1 }
3880     }
3881     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3882     \bool_lazy_or:nnT
3883         \l_@@_except_borders_bool
3884         { \clist_if_empty_p:N \l_@@_vlines_clist }
3885         { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3886     \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3887 }
3888 { \endNiceArray }

```

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

```

3889 \cs_new_protected:Npn \@@_NotEmpty:
3890 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

3891 \cs_set_protected:Npn \@@_renew_matrix:
3892 {
3893     \tl_map_inline:nn { pvVbB }
3894     { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3895 }

```

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

```

3896 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3897 {

```

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.

```

3898     \dim_compare:nNtT\l_@@_width_dim = \c_zero_dim
3899     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3900     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3901     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3902     \tl_if_empty:NF \l_@@_short_caption_tl
3903     {
3904         \tl_if_empty:NT \l_@@_caption_tl
3905         {
3906             \@@_error_or_warning:n { short-caption-without~caption }
3907             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3908         }
3909     }
3910     \tl_if_empty:NF \l_@@_label_tl
3911     {
3912         \tl_if_empty:NT \l_@@_caption_tl
3913         { \@@_error_or_warning:n { label~without~caption } }
3914     }

```

```

3915 \NewDocumentEnvironment { TabularNote } { b }
3916 {
3917   \bool_if:NTF \l_@@_in_code_after_bool
3918   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3919   {
3920     \tl_if_empty:NF \g_@@_tabularnote_tl
3921     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3922     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3923   }
3924 }
3925 { }
3926 \@@_settings_for_tabular:
3927 \NiceArray { #2 }
3928 }
3929 { \endNiceArray }
3930 \cs_new_protected:Npn \@@_settings_for_tabular:
3931 {
3932   \bool_set_true:N \l_@@_tabular_bool
3933   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3934   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3935   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3936 }

3937 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3938 {
3939   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3940   \dim_set:Nn \l_@@_width_dim { #1 }
3941   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3942   \@@_settings_for_tabular:
3943   \NiceArray { #3 }
3944 }
3945 {
3946   \endNiceArray
3947   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3948   { \@@_error:n { NiceTabularX~without~X } }
3949 }

3950 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3951 {
3952   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3953   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3954   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3955   \@@_settings_for_tabular:
3956   \NiceArray { #3 }
3957 }
3958 { \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`).

```

3959 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3960 {
3961   \bool_lazy_all:nT
3962   {
3963     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3964     { \l_@@_hvlines_bool }
3965     { ! \g_@@_delims_bool }

```

```

3966     { ! \l_@@_except_borders_bool }
3967   }
3968   {
3969     \bool_set_true:N \l_@@_except_borders_bool
3970     \clist_if_empty:NF \l_@@_corners_clist
3971     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3972     \tl_gput_right:Nn \g_@@_rules_tl
3973     {
3974       \@@_stroke_block:nnnnn
3975       {
3976         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3977         draw = \l_@@_rules_color_tl
3978       }
3979       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3980     }
3981   }
3982 }

3983 \cs_new_protected:Npn \@@_after_array:
3984 {

```

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`.

```

3985     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3986     \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.

```

3987     \bool_if:NT \g_@@_last_col_found_bool
3988     { \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`.

```

3989     \bool_if:NT \l_@@_last_col_without_value_bool
3990     { \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.

```

3991     \bool_if:NT \l_@@_last_row_without_value_bool
3992     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3993     \tl_gput_right:Ne \g_@@_aux_tl
3994     {
3995       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3996       {
3997         \int_use:N \l_@@_first_row_int ,
3998         \int_use:N \c@iRow ,
3999         \int_use:N \g_@@_row_total_int ,
4000         \int_use:N \l_@@_first_col_int ,
4001         \int_use:N \c@jCol ,
4002         \int_use:N \g_@@_col_total_int
4003       }
4004     }

4005     \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`).

```

4006   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4007   {
4008     \tl_gput_right:Ne \g_@@_aux_tl
4009     {
4010       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4011       { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4012     }
4013   }
4014   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4015   {
4016     \tl_gput_right:Ne \g_@@_aux_tl
4017     {
4018       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4019       { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4020       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4021       { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4022     }
4023   }

```

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

```

4024   \@@_create_diag_nodes:

```

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

```

4025   \pgfpicture
4026   \@@_create_aliases_last:
4027   \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4028   \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.

```

4029   \bool_if:NT \l_@@_parallelize_diags_bool
4030   {
4031     \int_gzero:N \g_@@_ddots_int
4032     \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.

```

4033     \dim_gzero:N \g_@@_delta_x_one_dim
4034     \dim_gzero:N \g_@@_delta_y_one_dim
4035     \dim_gzero:N \g_@@_delta_x_two_dim
4036     \dim_gzero:N \g_@@_delta_y_two_dim
4037   }
4038   \bool_set_false:N \l_@@_initial_open_bool
4039   \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.

```

4040   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4041   \@@_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.

```

4042   \clist_if_empty:NF \l_@@_corners_cells_clist
4043   {
4044     \bool_if:NTF \l_@@_no_cell_nodes_bool
4045     { \@@_error:n { corners-with-no-cell-nodes } }
4046     \@@_compute_corners:
4047   }

```

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`.

```

4048   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4049   \g_@@_pos_of_blocks_seq
4050   \g_@@_future_pos_of_blocks_seq
4051   \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-*}`).

```

4052   \@@_adjust_pos_of_blocks_seq:
4053   \@@_deal_with_rounded_corners:
4054   \legacy_if:nF { measuring@ } \@@_draw_rules:
4055   \tl_gclear:N \g_@@_rules_tl

```

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

```

4056   \IfPackageLoadedT { tikz }
4057   {
4058     \tikzset
4059     {
4060       every~picture / .style =
4061       {
4062         overlay ,
4063         remember~picture ,
4064         name~prefix = \@@_env: -
4065       }
4066     }
4067   }
4068   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4069   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4070   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4071   \cs_set_eq:NN \OverBrace \@@_OverBrace
4072   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4073   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4074   \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.

```

4075   \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4076   \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.

```

4077   \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).

```

4078   \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`).

```
4079 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4080 { \@@_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:`.

```
4081 \bool_set_true:N \l_@@_in_code_after_bool
4082 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4083 \scan_stop:
4084 \tl_gclear:N \g_nicematrix_code_after_tl
4085 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4086 \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.

```
4087 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4088 \tl_if_empty:NF \g_@@_pre_code_before_tl
4089 {
4090   \tl_gput_right:Ne \g_@@_aux_tl
4091   {
4092     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4093     { \exp_not:o \g_@@_pre_code_before_tl }
4094   }
4095   \tl_gclear:N \g_@@_pre_code_before_tl
4096 }
4097 \tl_if_empty:NF \g_nicematrix_code_before_tl
4098 {
4099   \tl_gput_right:Ne \g_@@_aux_tl
4100   {
4101     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4102     { \exp_not:o \g_nicematrix_code_before_tl }
4103   }
4104   \tl_gclear:N \g_nicematrix_code_before_tl
4105 }

4106 \str_gclear:N \g_@@_name_env_str
4107 \@@_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`.

```
4108 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4109 }

4110 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4111 {
4112   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4113   \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.

```
4114 \dim_set:Nn \l_@@_xdots_shorten_start_dim
4115 { 0.6 \l_@@_xdots_shorten_start_dim }
4116 \dim_set:Nn \l_@@_xdots_shorten_end_dim
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

4117     { 0.6 \l_@@_xdots_shorten_end_dim }
4118 }

```

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`.

```

4119 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4120 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4121 \cs_new_protected:Npn \@@_create_alias_nodes:
4122 {
4123   \int_step_inline:nn \c@iRow
4124   {
4125     \pgfnodealias
4126     { \l_@@_name_str - ##1 - last }
4127     { \@@_env: - ##1 - \int_use:N \c@jCol }
4128   }
4129   \int_step_inline:nn \c@jCol
4130   {
4131     \pgfnodealias
4132     { \l_@@_name_str - last - ##1 }
4133     { \@@_env: - \int_use:N \c@iRow - ##1 }
4134   }
4135   \pgfnodealias
4136   { \l_@@_name_str - last - last }
4137   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4138 }

```

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).

```

4139 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4140 {
4141   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4142   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4143 }

```

The following command must *not* be protected.

```

4144 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4145 {
4146   { #1 }
4147   { #2 }
4148   {
4149     \int_compare:nNnTF { #3 } > { 98 }
4150     { \int_use:N \c@iRow }
4151     { #3 }
4152   }
4153   {
4154     \int_compare:nNnTF { #4 } > { 98 }
4155     { \int_use:N \c@jCol }
4156     { #4 }
4157   }
4158   { #5 }
4159 }

```


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).

```

4160 \AtBeginDocument
4161 {
4162   \cs_new_protected:Npn \@@_draw_dotted_lines:
4163   {
4164     \c_@@_pgfortikzpicture_tl
4165     \@@_draw_dotted_lines_i:
4166     \c_@@_endpgfortikzpicture_tl
4167   }
4168 }

```

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

```

4169 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4170 {
4171   \pgfrememberpicturepositiononpagetrue
4172   \pgf@relevantforpicturesizefalse
4173   \g_@@_HVdotsfor_lines_tl
4174   \g_@@_Vdots_lines_tl
4175   \g_@@_Ddots_lines_tl
4176   \g_@@_Iddots_lines_tl
4177   \g_@@_Cdots_lines_tl
4178   \g_@@_Ldots_lines_tl
4179 }

4180 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4181 {
4182   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4183   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4184 }

```

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

```

4185 \pgfdeclareshape { @@_diag_node }
4186 {
4187   \savedanchor { \five }
4188   {
4189     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4190     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4191   }
4192   \anchor { 5 } { \five }
4193   \anchor { center } { \pgfpointorigin }
4194   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4195   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4196   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4197   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4198   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4199   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4200   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4201   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4202   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4203   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4204 }

```

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).

```

4205 \cs_new_protected:Npn \@@_create_diag_nodes:
4206 {
4207   \pgfpicture

```

```

4208 \pgfrememberpicturepositiononpagetrue
4209 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4210 {
4211     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4212     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4213     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4214     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4215     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4216     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4217     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4218     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4219     \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.

```

4220     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4221     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4222     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4223     \str_if_empty:NF \l_@@_name_str
4224     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4225 }

```

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

```

4226 \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4227 \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4228 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4229 \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4230 \pgfcoordinate
4231 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4232 \pgfnodealias
4233 { \@@_env: - last }
4234 { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4235 \str_if_empty:NF \l_@@_name_str
4236 {
4237     \pgfnodealias
4238     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4239     { \@@_env: - \int_use:N \l_tmpa_int }
4240     \pgfnodealias
4241     { \l_@@_name_str - last }
4242     { \@@_env: - last }
4243 }
4244 \endpgfpicture
4245 }

```

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:NTF \l_@@_initial_open_bool
{
\int_add:Nn \l_@@_initial_i_int { #3 }
\int_add:Nn \l_@@_initial_j_int { #4 }
\bool_set_true:N \l_@@_stop_loop_bool
}

```

```

    }
    {
      \cs_if_exist:cTF
      {
        @@ _ dotted _
        \int_use:N \l_@@_initial_i_int -
        \int_use:N \l_@@_initial_j_int
      }
      {
        \int_add:Nn \l_@@_initial_i_int { #3 }
        \int_add:Nn \l_@@_initial_j_int { #4 }
        \bool_set_true:N \l_@@_initial_open_bool
        \bool_set_true:N \l_@@_stop_loop_bool
      }
    }
    {
      \cs_if_exist:cTF
      {
        {
          pgf @ sh @ ns @ \@@_env:
          - \int_use:N \l_@@_initial_i_int
          - \int_use:N \l_@@_initial_j_int
        }
        { \bool_set_true:N \l_@@_stop_loop_bool }
        {
          \cs_set_nopar:cpn
          {
            @@ _ dotted _
            \int_use:N \l_@@_initial_i_int -
            \int_use:N \l_@@_initial_j_int
          }
          { }
        }
      }
    }
  }
}
\seq_gput_right:Ne \g_@@_pos_of_xdots_seq
{
  { \int_use:N \l_@@_initial_i_int }
  { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { \int_use:N \l_@@_final_i_int }
  { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
  { }
}
}

```

The following version is slightly more efficient.

```

4246 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4247 {

```

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

```

4248   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4249   \l_@@_initial_i_int = #1
4250   \l_@@_initial_j_int = #2
4251   \l_@@_final_i_int = #1
4252   \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.

```

4253   \let \l_@@_stop_loop_bool \c_false_bool
4254   \bool_do_until:Nn \l_@@_stop_loop_bool
4255   {

```

We test if we are still in the matrix.

```

4256      \advance \l_@@_final_i_int by #3
4257      \advance \l_@@_final_j_int by #4
4258      \let \l_@@_final_open_bool \c_false_bool
4259      \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4260        \if_int_compare:w #3 = \c_one_int
4261          \let \l_@@_final_open_bool \c_true_bool
4262        \else:
4263          \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4264            \let \l_@@_final_open_bool \c_true_bool
4265          \fi:
4266        \fi:
4267      \else:
4268        \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4269          \if_int_compare:w #4 = -1
4270            \let \l_@@_final_open_bool \c_true_bool
4271          \fi:
4272        \else:
4273          \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4274            \if_int_compare:w #4 = \c_one_int
4275              \let \l_@@_final_open_bool \c_true_bool
4276            \fi:
4277          \fi:
4278        \fi:
4279      \fi:
4280      \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.

```

4281      {

```

We do a step backwards.

```

4282      \advance \l_@@_final_i_int by - #3
4283      \advance \l_@@_final_j_int by - #4
4284      \let \l_@@_stop_loop_bool \c_true_bool
4285    }

```

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`.

```

4286      {
4287        \cs_if_exist:cTF
4288        {
4289          @@ _ dotted _
4290          \int_use:N \l_@@_final_i_int -
4291          \int_use:N \l_@@_final_j_int
4292        }
4293        {
4294          \advance \l_@@_final_i_int by - #3
4295          \advance \l_@@_final_j_int by - #4
4296          \let \l_@@_final_open_bool \c_true_bool
4297          \let \l_@@_stop_loop_bool \c_true_bool
4298        }
4299        {
4300          \cs_if_exist:cTF
4301          {
4302            pgf @ sh @ ns @ \@@_env:
4303            - \int_use:N \l_@@_final_i_int
4304            - \int_use:N \l_@@_final_j_int
4305          }
4306          { \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.

```

4307         {
4308             \cs_set_nopar:cpn
4309             {
4310                 @@ _ dotted _
4311                 \int_use:N \l_@@_final_i_int -
4312                 \int_use:N \l_@@_final_j_int
4313             }
4314             { }
4315         }
4316     }
4317 }
4318 }

```

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

```

4319     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4320     \l_tmpa_int = \l_@@_col_min_int
4321     \advance \l_tmpa_int by -1
4322     \bool_do_until:Nn \l_@@_stop_loop_bool
4323     {
4324         \advance \l_@@_initial_i_int by - #3
4325         \advance \l_@@_initial_j_int by - #4
4326         \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.

```

4327         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4328         \if_int_compare:w #3 = \c_one_int
4329         \let \l_@@_initial_open_bool \c_true_bool
4330     \else:

```

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

```

4331         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4332         \let \l_@@_initial_open_bool \c_true_bool
4333     \fi:
4334     \fi:
4335 \else:
4336     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4337     \if_int_compare:w #4 = \c_one_int
4338     \let \l_@@_initial_open_bool \c_true_bool
4339     \fi:
4340 \else:
4341     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4342     \if_int_compare:w #4 = -1
4343     \let \l_@@_initial_open_bool \c_true_bool
4344     \fi:
4345     \fi:
4346     \fi:
4347 \fi:
4348 \bool_if:NTF \l_@@_initial_open_bool
4349 {
4350     \advance \l_@@_initial_i_int by #3
4351     \advance \l_@@_initial_j_int by #4
4352     \let \l_@@_stop_loop_bool \c_true_bool
4353 }
4354 {
4355     \cs_if_exist:cTF
4356     {
4357         @@ _ dotted _
4358         \int_use:N \l_@@_initial_i_int -

```

```

4359         \int_use:N \l_@@_initial_j_int
4360     }
4361     {
4362         \advance \l_@@_initial_i_int by #3
4363         \advance \l_@@_initial_j_int by #4
4364         \let \l_@@_initial_open_bool \c_true_bool
4365         \let \l_@@_stop_loop_bool \c_true_bool
4366     }
4367     {
4368         \cs_if_exist:cTF
4369         {
4370             pgf @ sh @ ns @ \@@_env:
4371             - \int_use:N \l_@@_initial_i_int
4372             - \int_use:N \l_@@_initial_j_int
4373         }
4374         { \let \l_@@_stop_loop_bool \c_true_bool }
4375         {
4376             \cs_set_nopar:cpn
4377             {
4378                 @@ _ dotted _
4379                 \int_use:N \l_@@_initial_i_int -
4380                 \int_use:N \l_@@_initial_j_int
4381             }
4382             { }
4383         }
4384     }
4385 }
4386 }

```

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.

```

4387     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4388     {
4389         { \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`.

```

4390         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4391         { \int_use:N \l_@@_final_i_int }
4392         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4393         { }
4394     }
4395 }

```

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.

```

4396 \cs_new_protected:Npn \@@_open_shorten:
4397 {
4398     \bool_if:NT \l_@@_initial_open_bool
4399     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4400     \bool_if:NT \l_@@_final_open_bool
4401     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4402 }

```

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).

```

4403 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2

```



```

4404 {
4405   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4406   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4407   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4408   \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.

```

4409   \seq_if_empty:NTF \g_@@_submatrix_seq
4410   { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4411   {

```

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`.

```

4412     \seq_map_inline:Nn \g_@@_submatrix_seq
4413     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4414   }
4415 }

```

`#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.

```

4416 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4417 {
4418   \if_int_compare:w #3 > #1
4419   \else:
4420     \if_int_compare:w #1 > #5
4421     \else:
4422       \if_int_compare:w #4 > #2
4423       \else:
4424         \if_int_compare:w #2 > #6
4425         \else:
4426           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4427           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4428           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4429           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4430         \fi:
4431       \fi:
4432     \fi:
4433   \fi:
4434 }

4435 \cs_new_protected:Npn \@@_set_initial_coords:
4436 {
4437   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

4438 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4439 }
4440 \cs_new_protected:Npn \@@_set_final_coords:
4441 {
4442 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4443 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4444 }
4445 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4446 {
4447 \pgfpointanchor
4448 {
4449 \@@_env:
4450 - \int_use:N \l_@@_initial_i_int
4451 - \int_use:N \l_@@_initial_j_int
4452 }
4453 { #1 }
4454 \@@_set_initial_coords:
4455 }
4456 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4457 {
4458 \pgfpointanchor
4459 {
4460 \@@_env:
4461 - \int_use:N \l_@@_final_i_int
4462 - \int_use:N \l_@@_final_j_int
4463 }
4464 { #1 }
4465 \@@_set_final_coords:
4466 }
4467 \cs_new_protected:Npn \@@_open_x_initial_dim:
4468 {
4469 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4470 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4471 {
4472 \cs_if_exist:cT
4473 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4474 {
4475 \pgfpointanchor
4476 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4477 { west }
4478 \dim_set:Nn \l_@@_x_initial_dim
4479 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4480 }
4481 }

```

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

```

4482 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4483 {
4484 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4485 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4486 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4487 }
4488 }
4489 \cs_new_protected:Npn \@@_open_x_final_dim:
4490 {
4491 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4492 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4493 {
4494 \cs_if_exist:cT
4495 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4496 {
4497 \pgfpointanchor
4498 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }

```

```

4499         { east }
4500         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4501         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4502     }
4503 }

```

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

```

4504     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4505     {
4506         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4507         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4508         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4509     }
4510 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4511 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4512 {
4513     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4514     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4515     {
4516         \@@_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.

```

4517         \bool_if:NT \g_@@_aux_found_bool
4518         {
4519             {
4520                 \@@_open_shorten:
4521                 \int_if_zero:nTF { #1 }
4522                 { \color { nicematrix-first-row } }
4523             }

```

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.

```

4524         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4525         { \color { nicematrix-last-row } }
4526     }
4527     \keys_set:nn { nicematrix / xdots } { #3 }
4528     \@@_color:o \l_@@_xdots_color_tl
4529     \@@_actually_draw_Ldots:
4530 }
4531 }
4532 }
4533 }

```

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`.

```

4534 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4535 {
4536   \bool_if:NTF \l_@@_initial_open_bool
4537   {
4538     \@@_open_x_initial_dim:
4539     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4540     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4541   }
4542   { \@@_set_initial_coords_from_anchor:n { base-east } }
4543   \bool_if:NTF \l_@@_final_open_bool
4544   {
4545     \@@_open_x_final_dim:
4546     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4547     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4548   }
4549   { \@@_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.

```

4550   \bool_lazy_all:nTF
4551   {
4552     \l_@@_initial_open_bool
4553     \l_@@_final_open_bool
4554     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4555   }
4556   {
4557     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4558     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4559   }

```

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 (?).

```

4560   {
4561     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4562     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4563   }
4564   \@@_draw_line:
4565 }

```

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.

```

4566 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4567 {
4568   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4569   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4570   {
4571     \@@_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.

```

4572   \bool_if:NT \g_@@_aux_found_bool
4573   {
4574     {
4575       \@@_open_shorten:
4576       \int_if_zero:nTF { #1 }
4577       { \color { nicematrix-first-row } }
4578     }

```

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.

```

4579   \int_compare:nNnT { #1 } = \l_@@_last_row_int

```

```

4580         { \color { nicematrix-last-row } }
4581     }
4582     \keys_set:nn { nicematrix / xdots } { #3 }
4583     \@@_color:o \l_@@_xdots_color_tl
4584     \@@_actually_draw_Cdots:
4585 }
4586 }
4587 }
4588 }

```

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.`

```

4589 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4590 {
4591     \bool_if:NTF \l_@@_initial_open_bool
4592         \@@_open_x_initial_dim:
4593         { \@@_set_initial_coords_from_anchor:n { mid-east } }
4594     \bool_if:NTF \l_@@_final_open_bool
4595         \@@_open_x_final_dim:
4596         { \@@_set_final_coords_from_anchor:n { mid-west } }
4597     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4598     {
4599         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4600         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4601         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4602         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4603         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4604     }
4605     {
4606         \bool_if:NT \l_@@_initial_open_bool
4607         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4608         \bool_if:NT \l_@@_final_open_bool
4609         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4610     }
4611     \@@_draw_line:
4612 }
4613 \cs_new_protected:Npn \@@_open_y_initial_dim:
4614 {
4615     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4616     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4617     {
4618         \cs_if_exist:cT
4619         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4620         {
4621             \pgfpointanchor
4622             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4623             { north }
4624             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4625             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4626         }
4627     }
4628     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4629     {

```

```

4630 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4631 \dim_set:Nn \l_@@_y_initial_dim
4632 {
4633   \fp_to_dim:n
4634   {
4635     \pgf@y
4636     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4637   }
4638 }
4639 }
4640 }
4641 \cs_new_protected:Npn \@@_open_y_final_dim:
4642 {
4643   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4644   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4645   {
4646     \cs_if_exist:cT
4647     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4648     {
4649       \pgfpointanchor
4650       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4651       { south }
4652       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4653       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4654     }
4655   }
4656   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4657   {
4658     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4659     \dim_set:Nn \l_@@_y_final_dim
4660     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4661   }
4662 }

```

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.

```

4663 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4664 {
4665   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4666   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4667   {
4668     \@@_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.

```

4669   \bool_if:NT \g_@@_aux_found_bool
4670   {
4671     {
4672       \@@_open_shorten:
4673       \int_if_zero:nTF { #2 }
4674       { \color { nicematrix-first-col } }
4675       {
4676         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4677         { \color { nicematrix-last-col } }
4678       }
4679       \keys_set:nn { nicematrix / xdots } { #3 }
4680       \@@_color:o \l_@@_xdots_color_tl
4681       \bool_if:NTF \l_@@_Vbrace_bool
4682       \@@_actually_draw_Vbrace:
4683       \@@_actually_draw_Vdots:
4684     }
4685   }
4686 }
4687 }

```

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`.

```

4688 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4689 {
4690   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4691     \@@_actually_draw_Vdots_i:
4692     \@@_actually_draw_Vdots_ii:
4693   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4694   \@@_draw_line:
4695 }

```

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

```

4696 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4697 {
4698   \@@_open_y_initial_dim:
4699   \@@_open_y_final_dim:
4700   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4701 {
4702   \@@_qpoint:n { col - 1 }
4703   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4704   \dim_sub:Nn \l_@@_x_initial_dim
4705     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4706 }
4707 {
4708   \bool_lazy_and:nnTF
4709     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4710     { \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”.

```

4711 {
4712   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4713   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4714   \dim_add:Nn \l_@@_x_initial_dim
4715     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4716 }

```

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

```

4717 {
4718   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4719   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4720   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4721   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4722 }
4723 }
4724 }

```

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.

```

4725 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4726 {
4727   \bool_set_false:N \l_tmpa_bool
4728   \bool_if:NF \l_@@_initial_open_bool
4729   {
4730     \bool_if:NF \l_@@_final_open_bool
4731     {
4732       \@@_set_initial_coords_from_anchor:n { south-west }
4733       \@@_set_final_coords_from_anchor:n { north-west }
4734       \bool_set:Nn \l_tmpa_bool
4735       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4736     }
4737   }

```

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

```

4738   \bool_if:NTF \l_@@_initial_open_bool
4739   {
4740     \@@_open_y_initial_dim:
4741     \@@_set_final_coords_from_anchor:n { north }
4742     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4743   }
4744   {
4745     \@@_set_initial_coords_from_anchor:n { south }
4746     \bool_if:NTF \l_@@_final_open_bool
4747     \@@_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.

```

4748   {
4749     \@@_set_final_coords_from_anchor:n { north }
4750     \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4751     {
4752       \dim_set:Nn \l_@@_x_initial_dim
4753       {
4754         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4755         \l_@@_x_initial_dim \l_@@_x_final_dim
4756       }
4757     }
4758   }
4759 }
4760 }

```

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`.

```

4761 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4762 {
4763   \bool_if:NTF \l_@@_initial_open_bool
4764   \@@_open_y_initial_dim:
4765   { \@@_set_initial_coords_from_anchor:n { south } }
4766   \bool_if:NTF \l_@@_final_open_bool
4767   \@@_open_y_final_dim:
4768   { \@@_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.

```

4769 \int_if_zero:nTF \l_@@_initial_j_int
4770 {
4771   \@@_qpoint:n { col - 1 }
4772   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4773   \dim_sub:Nn \l_@@_x_initial_dim
4774   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4775 }

```

Elsewhere, the brace must be drawn left flush.

```

4776 {
4777   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4778   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4779   \dim_add:Nn \l_@@_x_initial_dim
4780   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4781 }

```

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

```

4782 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4783 \@@_draw_line:
4784 }

```

```

4785 \cs_new:Npn \@@_colsep:
4786 { \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.

```

4787 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4788 {
4789   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4790   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4791   {
4792     \@@_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.

```

4793   \bool_if:nT \g_@@_aux_found_bool
4794   {
4795     {
4796       \@@_open_shorten:
4797       \keys_set:nn { nicematrix / xdots } { #3 }
4798       \@@_color:o \l_@@_xdots_color_tl
4799       \@@_actually_draw_Ddots:
4800     }
4801   }
4802 }
4803 }

```

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.

```

4804 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4805 {
4806   \bool_if:NTF \l_@@_initial_open_bool
4807   {
4808     \@@_open_y_initial_dim:
4809     \@@_open_x_initial_dim:
4810   }
4811   { \@@_set_initial_coords_from_anchor:n { south-east } }
4812   \bool_if:NTF \l_@@_final_open_bool
4813   {
4814     \@@_open_x_final_dim:
4815     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4816   }
4817   { \@@_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.

```

4818   \bool_if:NT \l_@@_parallelize_diags_bool
4819   {
4820     \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).

```

4821     \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.

```

4822     {
4823       \dim_gset:Nn \g_@@_delta_x_one_dim
4824       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4825       \dim_gset:Nn \g_@@_delta_y_one_dim
4826       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4827     }

```

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.

```

4828     {
4829       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4830       {
4831         \dim_set:Nn \l_@@_y_final_dim
4832         {
4833           \l_@@_y_initial_dim +
4834           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4835           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4836         }
4837       }
4838     }
4839   }
4840   \@@_draw_line:
4841 }

```

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.

```

4842 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4843 {
4844   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4845   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4846   {
4847     \@@_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.

```

4848     \bool_if:NT \g_@@_aux_found_bool
4849     {
4850     {
4851         \@@_open_shorten:
4852         \keys_set:nn { nicematrix / xdots } { #3 }
4853         \@@_color:o \l_@@_xdots_color_tl
4854         \@@_actually_draw_Iddots:
4855     }
4856     }
4857 }
4858 }

```

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`.

```

4859 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4860 {
4861     \bool_if:NTF \l_@@_initial_open_bool
4862     {
4863         \@@_open_y_initial_dim:
4864         \@@_open_x_initial_dim:
4865     }
4866     { \@@_set_initial_coords_from_anchor:n { south-west } }
4867     \bool_if:NTF \l_@@_final_open_bool
4868     {
4869         \@@_open_y_final_dim:
4870         \@@_open_x_final_dim:
4871     }
4872     { \@@_set_final_coords_from_anchor:n { north-east } }
4873     \bool_if:NT \l_@@_parallelize_diags_bool
4874     {
4875         \int_gincr:N \g_@@_iddots_int
4876         \int_compare:nNnTF \g_@@_iddots_int = 1
4877         {
4878             \dim_gset:Nn \g_@@_delta_x_two_dim
4879             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4880             \dim_gset:Nn \g_@@_delta_y_two_dim
4881             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4882         }
4883         {
4884             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4885             {
4886                 \dim_set:Nn \l_@@_y_final_dim
4887                 {
4888                     \l_@@_y_initial_dim +
4889                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4890                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4891                 }
4892             }
4893         }
4894     }
4895     \@@_draw_line:
4896 }

```

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`

```

4897 \cs_new_protected:Npn \@@_draw_line:
4898 {
4899   \pgfrememberpicturepositiononpagetrue
4900   \pgf@relevantforpicturesizefalse
4901   \bool_lazy_or:nnTF
4902     \l_@@_dotted_bool
4903     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4904     \@@_draw_standard_dotted_line:
4905     \@@_draw_unstandard_dotted_line:
4906 }

```

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.

```

4907 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4908 {
4909   \begin { scope }
4910     \@@_draw_unstandard_dotted_line:o
4911     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4912 }

```

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.

```

4913 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4914 {
4915   \@@_draw_unstandard_dotted_line:nooo
4916   { #1 }
4917   \l_@@_xdots_up_tl
4918   \l_@@_xdots_down_tl
4919   \l_@@_xdots_middle_tl
4920 }
4921 \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.

```

4922 \AtBeginDocument
4923 {
4924   \IfPackageLoadedT { tikz }
4925   {
4926     \tikzset
4927     {
4928       @@_node_above / .style = { sloped , above } ,
4929       @@_node_below / .style = { sloped , below } ,
4930       @@_node_middle / .style =
4931       {

```

```

4932         sloped ,
4933         inner~sep = \c_@@_innersep_middle_dim
4934     }
4935 }
4936 }
4937 }

4938 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4939 {

```

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.

```

4940 \dim_zero_new:N \l_@@_l_dim
4941 \dim_set:Nn \l_@@_l_dim
4942 {
4943     \fp_to_dim:n
4944     {
4945         sqrt
4946         (
4947             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4948             +
4949             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4950         )
4951     }
4952 }

```

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.

```

4953 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4954 {
4955     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4956     \@@_draw_unstandard_dotted_line_i:
4957 }

```

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

```

4958 \bool_if:NT \l_@@_xdots_h_labels_bool
4959 {
4960     \tikzset
4961     {
4962         @@_node_above / .style = { auto = left } ,
4963         @@_node_below / .style = { auto = right } ,
4964         @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4965     }
4966 }
4967 \tl_if_empty:nF { #4 }
4968 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4969 \dim_zero:N \l_tmpa_dim
4970 \dim_zero:N \l_tmpb_dim
4971 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4972 {

```

We test whether the brace is vertical or horizontal.

```

4973     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4974     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4975     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4976 }
4977 {

```

```

4978 \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4979 {
4980 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4981 { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4982 { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4983 }
4984 }
4985 \use:e
4986 {
4987 \exp_not:N \begin { scope }
4988 [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
4989 }
4990 \draw
4991 [ #1 ]
4992 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4993 -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4994 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4995 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4996 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4997 \end { scope }
4998 \end { scope }
4999 }
5000 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5001 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5002 {
5003 \dim_set:Nn \l_tmpa_dim
5004 {
5005 \l_@@_x_initial_dim
5006 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5007 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5008 }
5009 \dim_set:Nn \l_tmpb_dim
5010 {
5011 \l_@@_y_initial_dim
5012 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5013 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5014 }
5015 \dim_set:Nn \l_@@_tmpc_dim
5016 {
5017 \l_@@_x_final_dim
5018 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5019 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5020 }
5021 \dim_set:Nn \l_@@_tmpd_dim
5022 {
5023 \l_@@_y_final_dim
5024 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5025 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5026 }
5027 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5028 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5029 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5030 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5031 }

```

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

```

5032 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5033 {
5034 {

```

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.

```

5035 \dim_zero_new:N \l_@@_l_dim
5036 \dim_set:Nn \l_@@_l_dim
5037 {
5038   \fp_to_dim:n
5039   {
5040     sqrt
5041     (
5042       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5043       +
5044       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5045     )
5046   }
5047 }

```

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.

```

5048 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5049 {
5050   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5051   \@@_draw_standard_dotted_line_i:
5052 }
5053 }
5054 \bool_lazy_all:nF
5055 {
5056   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5057   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5058   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5059 }
5060 \@@_labels_standard_dotted_line:
5061 }
5062 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5063 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5064 {

```

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

```

5065 \int_set:Nn \l_tmpa_int
5066 {
5067   \dim_ratio:nn
5068   {
5069     \l_@@_l_dim
5070     - \l_@@_xdots_shorten_start_dim
5071     - \l_@@_xdots_shorten_end_dim
5072   }
5073   \l_@@_xdots_inter_dim
5074 }

```

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

```

5075 \dim_set:Nn \l_tmpa_dim
5076 {
5077   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5078   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5079 }
5080 \dim_set:Nn \l_tmpb_dim
5081 {
5082   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5083   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5084 }

```

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.

```

5085 \dim_gadd:Nn \l_@@_x_initial_dim
5086 {
5087   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5088   \dim_ratio:nn
5089   {
5090     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5091     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5092   }
5093   { 2 \l_@@_l_dim }
5094 }
5095 \dim_gadd:Nn \l_@@_y_initial_dim
5096 {
5097   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5098   \dim_ratio:nn
5099   {
5100     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5101     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5102   }
5103   { 2 \l_@@_l_dim }
5104 }
5105 \pgf@relevantforpicturesizefalse
5106 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5107 {
5108   \pgfpathcircle
5109   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5110   \l_@@_xdots_radius_dim
5111   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5112   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5113 }
5114 \pgfusepathqfill
5115 }

5116 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5117 {
5118   \pgfscope
5119   \pgftransformshift
5120   {
5121     \pgfpointlineattime { 0.5 }
5122     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5123     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5124   }
5125   \fp_set:Nn \l_tmpa_fp
5126   {
5127     atand
5128     (
5129       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5130       \l_@@_x_final_dim - \l_@@_x_initial_dim
5131     )
5132   }
5133   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5134   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5135   \tl_if_empty:NF \l_@@_xdots_middle_tl
5136   {
5137     \begin { pgfscope }
5138     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5139     \pgfnode
5140     { rectangle }
5141     { center }
5142     {
5143       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }

```



```

5144         {
5145             $ % $
5146             \scriptstyle \l_@@_xdots_middle_tl
5147             $ % $
5148         }
5149     }
5150     { }
5151     {
5152         \pgfsetfillcolor { white }
5153         \pgfusepath { fill }
5154     }
5155     \end { pgfscope }
5156 }
5157 \tl_if_empty:NF \l_@@_xdots_up_tl
5158 {
5159     \pgfnode
5160     { rectangle }
5161     { south }
5162     {
5163         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5164         {
5165             $ % $
5166             \scriptstyle \l_@@_xdots_up_tl
5167             $ % $
5168         }
5169     }
5170     { }
5171     { \pgfusepath { } }
5172 }
5173 \tl_if_empty:NF \l_@@_xdots_down_tl
5174 {
5175     \pgfnode
5176     { rectangle }
5177     { north }
5178     {
5179         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5180         {
5181             $ % $
5182             \scriptstyle \l_@@_xdots_down_tl
5183             $ % $
5184         }
5185     }
5186     { }
5187     { \pgfusepath { } }
5188 }
5189 \endpgfscope
5190 }

```

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.

```

5191 \AtBeginDocument
5192 {

```

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

```

5193 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5194 \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5195 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5196 {
5197   \int_if_zero:nTF \c@jCol
5198   { \@@_error:nn { in~first~col } { \Ldots } }
5199   {
5200     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5201     { \@@_error:nn { in~last~col } { \Ldots } }
5202     {
5203       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5204       { #1 , down = #2 , up = #3 , middle = #4 }
5205     }
5206   }
5207   \bool_if:NF \l_@@_nullify_dots_bool
5208   { \phantom { \ensuremath { \@@_old_ldots: } } }
5209   \bool_gset_true:N \g_@@_empty_cell_bool
5210 }

```

```

5211 \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5212 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5213 {
5214   \int_if_zero:nTF \c@jCol
5215   { \@@_error:nn { in~first~col } { \Cdots } }
5216   {
5217     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5218     { \@@_error:nn { in~last~col } { \Cdots } }
5219     {
5220       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5221       { #1 , down = #2 , up = #3 , middle = #4 }
5222     }
5223   }
5224   \bool_if:NF \l_@@_nullify_dots_bool
5225   { \phantom { \ensuremath { \@@_old_cdots: } } }
5226   \bool_gset_true:N \g_@@_empty_cell_bool
5227 }

```

```

5228 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5229 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5230 {
5231   \int_if_zero:nTF \c@iRow
5232   { \@@_error:nn { in~first~row } { \Vdots } }
5233   {
5234     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5235     { \@@_error:nn { in~last~row } { \Vdots } }
5236     {
5237       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5238       { #1 , down = #2 , up = #3 , middle = #4 }
5239     }
5240   }
5241   \bool_if:NF \l_@@_nullify_dots_bool
5242   { \phantom { \ensuremath { \@@_old_vdots: } } }
5243   \bool_gset_true:N \g_@@_empty_cell_bool
5244 }

```

```

5245 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }

```

```

5246 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5247 {
5248   \int_case:nnF \c@iRow
5249   {
5250     0 { \@@_error:nn { in-first~row } { \Ddots } }
5251     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Ddots } }
5252   }
5253   {
5254     \int_case:nnF \c@jCol
5255     {
5256       0 { \@@_error:nn { in-first~col } { \Ddots } }
5257       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Ddots } }
5258     }
5259     {
5260       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5261       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5262       { #1 , down = #2 , up = #3 , middle = #4 }
5263     }
5264   }
5265 }
5266 \bool_if:NF \l_@@_nullify_dots_bool
5267 { \phantom { \ensuremath { \@@_old_ddots: } } }
5268 \bool_gset_true:N \g_@@_empty_cell_bool
5269 }

5270 \cs_new_protected:Npn \@@_Iddots:
5271 { \@@_collect_options:n { \@@_Iddots_i } }
5272 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5273 {
5274   \int_case:nnF \c@iRow
5275   {
5276     0 { \@@_error:nn { in-first~row } { \Iddots } }
5277     \l_@@_last_row_int { \@@_error:nn { in-last~row } { \Iddots } }
5278   }
5279   {
5280     \int_case:nnF \c@jCol
5281     {
5282       0 { \@@_error:nn { in-first~col } { \Iddots } }
5283       \l_@@_last_col_int { \@@_error:nn { in-last~col } { \Iddots } }
5284     }
5285     {
5286       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5287       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5288       { #1 , down = #2 , up = #3 , middle = #4 }
5289     }
5290   }
5291   \bool_if:NF \l_@@_nullify_dots_bool
5292   { \phantom { \ensuremath { \@@_old_iddots: } } }
5293   \bool_gset_true:N \g_@@_empty_cell_bool
5294 }
5295 }

```

End of the \AddToHook.

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

```

5296 \keys_define:nn { nicematrix / Ddots }
5297 {
5298   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5299   draw-first .value_forbidden:n = true ,
5300 }

```

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

```

5301 \cs_new_protected:Npn \@@_Hspace:
5302 {
5303   \bool_gset_true:N \g_@@_empty_cell_bool
5304   \hspace
5305 }

```

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`.

```

5306 \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`.

```

5307 \cs_new:Npn \@@_Hdotsfor:
5308 {
5309   \bool_lazy_and:nnTF
5310     { \int_if_zero_p:n \c@jCol }
5311     { \int_if_zero_p:n \l_@@_first_col_int }
5312     {
5313       \bool_if:NTF \g_@@_after_col_zero_bool
5314       {
5315         \multicolumn { 1 } { c } { }
5316         \@@_Hdotsfor_i:
5317       }
5318       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5319     }
5320   {
5321     \multicolumn { 1 } { c } { }
5322     \@@_Hdotsfor_i:
5323   }
5324 }

```

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:`).

```

5325 \AtBeginDocument
5326 {

```

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

```

5327   \cs_new_protected:Npn \@@_Hdotsfor_i:
5328     { \@@_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`).

```

5329   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5330   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5331     {
5332       \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5333       {
5334         \@@_Hdotsfor:nnnn
5335         { \int_use:N \c@iRow }
5336         { \int_use:N \c@jCol }
5337         { #2 }
5338         {
5339           #1 , #3 ,
5340           down = \exp_not:n { #4 } ,
5341           up = \exp_not:n { #5 } ,
5342           middle = \exp_not:n { #6 }
5343         }
5344       }
5345       \prg_replicate:nn { #2 - 1 }

```

```

5346     {
5347     &
5348     \multicolumn { 1 } { c } { }
5349     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5350     }
5351   }
5352 }

```

```

5353 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5354 {
5355   \bool_set_false:N \l_@@_initial_open_bool
5356   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5357   \int_set:Nn \l_@@_initial_i_int { #1 }
5358   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5359   \int_compare:nNnTF { #2 } = 1
5360   {
5361     \int_set:Nn \l_@@_initial_j_int 1
5362     \bool_set_true:N \l_@@_initial_open_bool
5363   }
5364   {
5365     \cs_if_exist:cTF
5366     {
5367       pgf @ sh @ ns @ \@@_env:
5368       - \int_use:N \l_@@_initial_i_int
5369       - \int_eval:n { #2 - 1 }
5370     }
5371     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5372     {
5373       \int_set:Nn \l_@@_initial_j_int { #2 }
5374       \bool_set_true:N \l_@@_initial_open_bool
5375     }
5376   }
5377   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5378   {
5379     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5380     \bool_set_true:N \l_@@_final_open_bool
5381   }
5382   {
5383     \cs_if_exist:cTF
5384     {
5385       pgf @ sh @ ns @ \@@_env:
5386       - \int_use:N \l_@@_final_i_int
5387       - \int_eval:n { #2 + #3 }
5388     }
5389     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5390     {
5391       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5392       \bool_set_true:N \l_@@_final_open_bool
5393     }
5394   }
5395   \bool_if:NT \g_@@_aux_found_bool
5396   {
5397     {
5398       \@@_open_shorten:
5399       \int_if_zero:nTF { #1 }
5400       { \color { nicematrix-first-row } }
5401       {
5402         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5403         { \color { nicematrix-last-row } }
5404       }

```

```

5405         \keys_set:nn { nicematrix / xdots } { #4 }
5406         \@@_color:o \l_@@_xdots_color_tl
5407         \@@_actually_draw_Ldots:
5408     }
5409 }

```

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).

```

5410     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5411     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - #1 } { } }
5412 }

```

```

5413 \AtBeginDocument
5414 {
5415     \cs_new_protected:Npn \@@_Vdotsfor:
5416     { \@@_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`).

```

5417     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5418     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5419     {
5420         \bool_gset_true:N \g_@@_empty_cell_bool
5421         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5422         {
5423             \@@_Vdotsfor:nnnn
5424             { \int_use:N \c@iRow }
5425             { \int_use:N \c@jCol }
5426             { #2 }
5427             {
5428                 #1 , #3 ,
5429                 down = \exp_not:n { #4 } ,
5430                 up = \exp_not:n { #5 } ,
5431                 middle = \exp_not:n { #6 }
5432             }
5433         }
5434     }
5435 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5436 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5437 {
5438     \bool_set_false:N \l_@@_initial_open_bool
5439     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5440     \int_set:Nn \l_@@_initial_j_int { #2 }
5441     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5442     \int_compare:nNnTF { #1 } = 1
5443     {
5444         \int_set:Nn \l_@@_initial_i_int 1
5445         \bool_set_true:N \l_@@_initial_open_bool
5446     }
5447     {
5448         \cs_if_exist:cTF
5449         {
5450             pgf @ sh @ ns @ \@@_env:

```

```

5451         - \int_eval:n { #1 - 1 }
5452         - \int_use:N \l_@@_initial_j_int
5453     }
5454     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5455     {
5456         \int_set:Nn \l_@@_initial_i_int { #1 }
5457         \bool_set_true:N \l_@@_initial_open_bool
5458     }
5459 }
5460 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5461 {
5462     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5463     \bool_set_true:N \l_@@_final_open_bool
5464 }
5465 {
5466     \cs_if_exist:cTF
5467     {
5468         pgf @ sh @ ns @ \@@_env:
5469         - \int_eval:n { #1 + #3 }
5470         - \int_use:N \l_@@_final_j_int
5471     }
5472     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5473     {
5474         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5475         \bool_set_true:N \l_@@_final_open_bool
5476     }
5477 }
5478 \bool_if:NT \g_@@_aux_found_bool
5479 {
5480     {
5481         \@@_open_shorten:
5482         \int_if_zero:nTF { #2 }
5483         { \color { nicematrix-first-col } }
5484         {
5485             \int_compare:nNnT { #2 } = \g_@@_col_total_int
5486             { \color { nicematrix-last-col } }
5487         }
5488         \keys_set:nn { nicematrix / xdots } { #4 }
5489         \@@_color:o \l_@@_xdots_color_tl
5490         \bool_if:NTF \l_@@_Vbrace_bool
5491         \@@_actually_draw_Vbrace:
5492         \@@_actually_draw_Vdots:
5493     }
5494 }

```

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).

```

5495     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5496     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5497 }

```

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

```

5498 \NewDocumentCommand \@@_rotate: { 0 { } }
5499 {
5500     \bool_gset_true:N \g_@@_rotate_bool
5501     \keys_set:nn { nicematrix / rotate } { #1 }
5502     \ignorespaces
5503 }

```

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

```

5504 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5505 \keys_define:nn { nicematrix / rotate }
5506 {
5507   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5508   c .value_forbidden:n = true ,
5509   -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5510   -90 .value_forbidden:n = true ,
5511   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5512 }

```

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

```

5513 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5514 {
5515   \tl_if_empty:nTF { #2 }
5516     { #1 }
5517     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5518 }
5519 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5520 { \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).

```

5521 \AtBeginDocument
5522 {

```

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

```

5523   \tl_set_rescan:Nnn \l_tmpa_tl { }
5524   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5525   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5526   {
5527     {
5528       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5529       \@@_color:o \l_@@_xdots_color_tl
5530       \use:e
5531       {
5532         \@@_line_i:nn
5533         { \@@_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`.


```

5534         { \@@_double_int_eval:n #3 - \q_stop }
5535     }
5536 }
5537 }
5538 }
5539 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5540 {
5541     \bool_set_false:N \l_@@_initial_open_bool
5542     \bool_set_false:N \l_@@_final_open_bool
5543     \bool_lazy_or:nnTF
5544     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5545     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5546     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

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

```

5547     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5548 }
5549 \AtBeginDocument
5550 {
5551     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5552     {

```

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:`.

```

5553     \c_@@_pgfortikzpicture_tl
5554     \@@_draw_line_iii:nn { #1 } { #2 }
5555     \c_@@_endpgfortikzpicture_tl
5556 }
5557 }

```

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

```

5558 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5559 {
5560     \pgfrememberpicturepositiononpagetrue
5561     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5562     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5563     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5564     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5565     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5566     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5567     \@@_draw_line:
5568 }

```

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.

```
5569 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5570 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }
```

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

```
5571 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5572 {
5573   \tl_gput_right:Ne \g_@@_row_style_tl
5574 }
```

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

```
5575   \exp_not:N
5576   \@@_if_row_less_than:nn
5577   { \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).

```
5578   {
5579     \exp_not:N
5580     \@@_if_col_greater_than:nn
5581     { \int_use:N \c@jCol }
5582     { \exp_not:n { #1 } \scan_stop: }
5583   }
5584 }
5585 }
5586 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5587 \keys_define:nn { nicematrix / RowStyle }
5588 {
5589   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5590   cell-space-top-limit+ .code:n =
5591     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5592   cell-space-top-limit+ .value_required:n = true ,
5593   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5594   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5595   cell-space-bottom-limit+ .code:n =
5596     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5597   cell-space-bottom-limit+ .value_required:n = true ,
5598   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5599   cell-space-limits .meta:n =
5600   {
5601     cell-space-top-limit = #1 ,
5602     cell-space-bottom-limit = #1 ,
5603   } ,
5604   cell-space-limits+ .meta:n =
5605   {
5606     cell-space-top-limit += #1 ,
5607     cell-space-bottom-limit += #1 ,
5608   } ,
5609   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5610   color .tl_set:N = \l_@@_color_tl ,
5611   color .value_required:n = true ,
5612   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5613   nb-rows .code:n =
5614     \str_if_eq:eeTF { #1 } { * }
5615     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5616     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5617   nb-rows .value_required:n = true ,
5618   fill .tl_set:N = \l_@@_fill_tl ,
5619   fill .value_required:n = true ,
```

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

```

5620     opacity .tl_set:N = \l_@@_opacity_tl ,
5621     opacity .value_required:n = true ,
5622     rowcolor .tl_set:N = \l_@@_fill_tl ,
5623     rowcolor .value_required:n = true ,
5624     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5625     rounded-corners .default:n = 4 pt ,
5626     unknown .code:n =
5627       \@@_unknown_key:nn
5628       { nicematrix / RowStyle }
5629       { Unknown-key-for-RowStyle }
5630   }

```

```

5631 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5632 {
5633   \group_begin:
5634   \tl_clear:N \l_@@_fill_tl
5635   \tl_clear:N \l_@@_opacity_tl
5636   \tl_clear:N \l_@@_color_tl
5637   \int_set:Nn \l_@@_key_nb_rows_int 1
5638   \dim_zero:N \l_@@_rounded_corners_dim
5639   \dim_zero:N \l_tmpa_dim
5640   \dim_zero:N \l_tmpb_dim
5641   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5642   \tl_if_empty:NF \l_@@_fill_tl
5643   {
5644     \@@_add_opacity_to_fill:
5645     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5646     {

```

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

```

5647       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5648       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5649       {
5650         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5651         - *
5652       }
5653       { \dim_use:N \l_@@_rounded_corners_dim }
5654     }
5655   }
5656   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5657   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5658   {
5659     \@@_put_in_row_style:e
5660     {
5661       \@@_put_in_cell_after_hook:n
5662       {

```

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

```

5663       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5664       { \dim_use:N \l_tmpa_dim }
5665     }
5666   }
5667 }

```

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

```

5668   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5669   {
5670     \@@_put_in_row_style:e
5671     {

```

```

5672         \@@_put_in_cell_after_hook:n
5673         {
5674             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5675             { \dim_use:N \l_tmpb_dim }
5676         }
5677     }
5678 }

```

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

```

5679     \tl_if_empty:NF \l_@@_color_tl
5680     {
5681         \@@_put_in_row_style:e
5682         {
5683             \mode_leave_vertical:
5684             \@@_color:n { \l_@@_color_tl }
5685         }
5686     }

```

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

```

5687     \bool_if:NT \l_@@_bold_row_style_bool
5688     {
5689         \@@_put_in_row_style:n
5690         {
5691             \exp_not:n
5692             {
5693                 \if_mode_math:
5694                 $ % $
5695                 \bfseries \boldmath
5696                 $ % $
5697             \else:
5698                 \bfseries \boldmath
5699             \fi:
5700         }
5701     }
5702 }
5703 \group_end:
5704 \g_@@_row_style_tl
5705 \ignorespaces
5706 }

```

The following commande must *not* be protected.

```

5707 \cs_new:Npn \@@_rounded_from_row:n #1
5708 {
5709     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5710     { \int_eval:n { #1 } - 1 }
5711     {
5712         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5713         - \exp_not:n { \int_use:N \c@jCol }
5714     }
5715     { \dim_use:N \l_@@_rounded_corners_dim }
5716 }

```

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).

```
5717 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5718 {
```

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.

```
5719 \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`.

```
5720 \str_if_in:nnF { #1 } { !! }
5721 {
5722 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5723 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5724 }
5725 \int_if_zero:nTF \l_tmpa_int
```

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

```
5726 {
5727 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5728 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5729 }
```

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

```
5730 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5731 }
5732 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
5733 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5734 {
5735 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5736 {
```

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

```
5737 \group_begin:
5738 \pgfsetcornersarced
5739 { \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.

```
5740 \bool_if:NTF \l_@@_hvlines_bool
5741 {
```

```

5742     \pgfpathrectanglecorners
5743     {
5744         \pgfpointadd
5745         { \@@_qpoint:n { row-1 } }
5746         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5747     }
5748     {
5749         \pgfpointadd
5750         {
5751             \@@_qpoint:n
5752             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5753         }
5754         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5755     }
5756 }
5757 {
5758     \pgfpathrectanglecorners
5759     { \@@_qpoint:n { row-1 } }
5760     {
5761         \pgfpointadd
5762         {
5763             \@@_qpoint:n
5764             { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5765         }
5766         { \pgfpoint \c_zero_dim \arrayrulewidth }
5767     }
5768 }
5769 \pgfusepath { clip }
5770 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5771     }
5772 }

```

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

```

5773 \cs_new_protected:Npn \@@_actually_color:
5774 {
5775     \pgfpicture
5776     \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.

```

5777     \@@_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).

```

5778     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5779     {
5780         \int_compare:nNnTF { ##1 } = 1
5781         {
5782             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5783             \use:c { g_@@_color _ 1 _tl }
5784             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5785         }
5786         {
5787             \pgfscope
5788             \@@_color_opacity: ##2
5789             \use:c { g_@@_color _ ##1 _tl }
5790             \tl_gclear:c { g_@@_color _ ##1 _tl }
5791             \pgfusepath { fill }
5792             \endpgfscope
5793         }
5794     }

```

```

5795 \endpgfpicture
5796 }

```

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

```

5797 \cs_new_protected:Npn \@@_color_opacity:
5798 {
5799   \peek_meaning:NTF [
5800     \@@_color_opacity:w
5801     { \@@_color_opacity:w [ ] }
5802   }

```

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.

```

5803 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5804 {
5805   \tl_clear:N \l_tmpa_tl
5806   \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.

```

5807   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5808   \tl_if_empty:NTF \l_tmpb_tl
5809     \@declaredcolor
5810     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5811 }

```

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

```

5812 \keys_define:nn { nicematrix / color-opacity }
5813 {
5814   opacity .tl_set:N          = \l_tmpa_tl ,
5815   opacity .value_required:n = true
5816 }

```

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

```

5817 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5818 {
5819   \def \l_@@_rows_tl { #1 }
5820   \def \l_@@_cols_tl { #2 }
5821   \@@_cartesian_path:
5822 }

```

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

```

5823 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5824 {
5825   \tl_if_blank:nF { #2 }
5826   {
5827     \@@_add_to_colors_seq:en
5828     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5829     { \@@_cartesian_color:nn { #3 } { - } }
5830   }
5831 }

```

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

```

5832 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5833 {
5834   \tl_if_blank:nF { #2 }
5835   {
5836     \@@_add_to_colors_seq:en
5837     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }

```

```

5838         { \@@_cartesian_color:nn { - } { #3 } }
5839     }
5840 }

```

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

```

5841 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5842 {
5843     \tl_if_blank:nF { #2 }
5844     {
5845         \@@_add_to_colors_seq:en
5846         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5847         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5848     }
5849 }

```

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

```

5850 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5851 {
5852     \tl_if_blank:nF { #2 }
5853     {
5854         \@@_add_to_colors_seq:en
5855         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5856         { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5857     }
5858 }

```

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

```

5859 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5860 {
5861     \@@_cut_on_hyphen:w #1 \q_stop
5862     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5863     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5864     \@@_cut_on_hyphen:w #2 \q_stop
5865     \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5866     \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`.

```

5867     \@@_cartesian_path:n { #3 }
5868 }

```

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

```

5869 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5870 {
5871     \clist_map_inline:nn { #3 }
5872     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5873 }

```

```

5874 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5875 {
5876     \int_step_inline:nn \c@iRow
5877     {
5878         \int_step_inline:nn \c@jCol
5879         {
5880             \int_if_even:nTF { ####1 + ##1 }
5881             { \@@_cellcolor [ #1 ] { #2 } }
5882             { \@@_cellcolor [ #1 ] { #3 } }
5883             { ##1 - ####1 }
5884         }
5885     }
5886 }

```


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”.

```

5887 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5888 {
5889   \@@_rectanglecolor [ #1 ] { #2 }
5890   { 1 - 1 }
5891   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5892 }

5893 \keys_define:nn { nicematrix / rowcolors }
5894 {
5895   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5896   cols .tl_set:N = \l_@@_cols_tl ,
5897   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5898   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5899 }

```

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*.

```

5900 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5901 {

```

`\l_@@_colors_seq` will be the list of colors.

```

5902   \seq_clear_new:N \l_@@_colors_seq
5903   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5904   \tl_clear_new:N \l_@@_cols_tl
5905   \tl_set:Nn \l_@@_cols_tl { - }
5906   \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).

```

5907   \int_zero_new:N \l_@@_color_int
5908   \int_set:Nn \l_@@_color_int 1
5909   \bool_if:NT \l_@@_respect_blocks_bool
5910   {

```

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`).

```

5911     % modified 2026-02-18
5912     \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5913     { \@@_not_in_exterior_p:nnnnn ##1 }
5914   }

```

#2 is the list of intervals of rows.

```

5915   \clist_map_inline:nn { #2 }
5916   {
5917     \tl_set:Nn \l_tmpa_tl { ##1 }
5918     \tl_if_in:NnTF \l_tmpa_tl { - }
5919     { \@@_cut_on_hyphen:w ##1 \q_stop }
5920     { \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.

```

5921     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5922     \int_set:Nn \l_@@_color_int
5923     { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5924     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5925     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5926     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
5927 \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).

```
5928 \bool_if:NT \l_@@_respect_blocks_bool
5929 {
5930   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5931   { \@@_intersect_our_row_p:nnnnn ###1 }
5932   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }
```

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

```
5933 }
5934 \tl_set:Ne \l_@@_rows_tl
5935 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
5936 \tl_set:Ne \l_@@_color_tl
5937 {
5938   \@@_color_index:n
5939   {
5940     \int_mod:nn
5941     { \l_@@_color_int - 1 }
5942     { \seq_count:N \l_@@_colors_seq }
5943     + 1
5944   }
5945 }
5946 \tl_if_empty:NF \l_@@_color_tl
5947 {
5948   \use:e
5949   {
5950     \@@_add_to_colors_seq:nn
5951     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5952     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5953   }
5954 }
5955 \int_incr:N \l_@@_color_int
5956 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5957 }
5958 }
5959 }
```

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.

```
5960 \cs_new:Npn \@@_color_index:n #1
5961 {
```

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

```
5962 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5963 { \@@_color_index:n { #1 - 1 } }
5964 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5965 }
```

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.

```
5966 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5967 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

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

```
5968 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5969 {
5970   \int_compare:nNnT { #3 } > \l_tmpb_int
5971   { \int_set:Nn \l_tmpb_int { #3 } }
5972 }
```

```

5973 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5974 {
5975   \int_if_zero:nTF { #4 }
5976   \prg_return_false:
5977   {
5978     \int_compare:nNnTF { #2 } > \c@jCol
5979     \prg_return_false:
5980     \prg_return_true:
5981   }
5982 }

```

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

```

5983 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5984 {
5985   \int_compare:nNnTF { #1 } > \l_tmpa_int
5986   \prg_return_false:
5987   {
5988     \int_compare:nNnTF \l_tmpa_int > { #3 }
5989     \prg_return_false:
5990     \prg_return_true:
5991   }
5992 }

```

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).

```

5993 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5994 {
5995   \dim_compare:nNnTF { #1 } = \c_zero_dim
5996   {
5997     \bool_if:NTF \l_@@_nocolor_used_bool
5998     { \@@_cartesian_path_normal_ii: }
5999     {
6000       \clist_if_empty:NTF \l_@@_corners_cells_clist
6001       { \@@_cartesian_path_normal_i:n { #1 } }
6002       { \@@_cartesian_path_normal_ii: }
6003     }
6004   }
6005   { \@@_cartesian_path_normal_i:n { #1 } }
6006 }

```

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.

```

6007 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6008 {
6009   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6010   \clist_map_inline:Nn \l_@@_cols_tl
6011   {

```

We use \def instead of \tl_set:Nn for efficiency only.

```

6012     \def \l_tmpa_tl { ##1 }
6013     \tl_if_in:NnTF \l_tmpa_tl { - }
6014     { \@@_cut_on_hyphen:w ##1 \q_stop }
6015     { \def \l_tmpb_tl { ##1 } }
6016     \tl_if_empty:NTF \l_tmpa_tl
6017     { \def \l_tmpa_tl { 1 } }
6018     {

```

```

6019     \str_if_eq:eeT \l_tmpa_tl { * }
6020     { \def \l_tmpa_tl { 1 } }
6021 }
6022 \int_compare:nNtT \l_tmpa_tl > \g_@@_col_total_int
6023 { \@@_error:n { Invalid~col~number } }
6024 \tl_if_empty:NTF \l_tmpb_tl
6025 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6026 {
6027     \str_if_eq:eeT \l_tmpb_tl { * }
6028     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6029 }
6030 \int_compare:nNtT \l_tmpb_tl > \g_@@_col_total_int
6031 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

6032 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6033 \@@_qpoint:n { col - \l_tmpa_tl }
6034 \int_compare:nNtTF \l_@@_first_col_int = \l_tmpa_tl
6035 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6036 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6037 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6038 \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.

```

6039 \clist_map_inline:Nn \l_@@_rows_tl
6040 {
6041     \def \l_tmpa_tl { #####1 }
6042     \tl_if_in:NnTF \l_tmpa_tl { - }
6043     { \@@_cut_on_hyphen:w #####1 \q_stop }
6044     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6045     \tl_if_empty:NTF \l_tmpa_tl
6046     { \def \l_tmpa_tl { 1 } }
6047     {
6048         \str_if_eq:eeT \l_tmpa_tl { * }
6049         { \def \l_tmpa_tl { 1 } }
6050     }
6051     \tl_if_empty:NTF \l_tmpb_tl
6052     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6053     {
6054         \str_if_eq:eeT \l_tmpb_tl { * }
6055         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6056     }
6057     \int_compare:nNtT \l_tmpa_tl > \g_@@_row_total_int
6058     { \@@_error:n { Invalid~row~number } }
6059     \int_compare:nNtT \l_tmpb_tl > \g_@@_row_total_int
6060     { \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`.

```

6061 \cs_if_exist:cF
6062 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6063 {
6064     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6065     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6066     \@@_qpoint:n { row - \l_tmpa_tl }
6067     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6068     \pgfpathrectanglecorners
6069     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6070     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6071 }
6072 }
6073 }
6074 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6075 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6076 {
6077   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6078   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6079   \clist_map_inline:Nn \l_@@_cols_tl
6080   {
6081     \@@_qpoint:n { col - ##1 }
6082     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6083     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6084     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6085     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6086     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6087     \clist_map_inline:Nn \l_@@_rows_tl
6088     {
6089       \@@_if_in_corner:nF { #####1 - ##1 }
6090       {
6091         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6092         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6093         \@@_qpoint:n { row - #####1 }
6094         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6095         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6096         {
6097           \pgfpathrectanglecorners
6098           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6099           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6100         }
6101       }
6102     }
6103   }
6104 }

```

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`).

```

6105 \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

```

6106 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6107 {
6108   \bool_set_true:N \l_@@_nocolor_used_bool
6109   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6110   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6111   \clist_map_inline:Nn \l_@@_rows_tl
6112   {
6113     \clist_map_inline:Nn \l_@@_cols_tl
6114     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6115   }
6116 }

```

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.

```

6117 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6118 {
6119   \clist_set_eq:NN \l_tmpa_clist #1

```

```

6120 \clist_clear:N #1
6121 \clist_map_inline:Nn \l_tmpa_clist
6122 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6123 \def \l_tmpa_tl { ##1 }
6124 \tl_if_in:NnTF \l_tmpa_tl { - }
6125 { \@@_cut_on_hyphen:w ##1 \q_stop }
6126 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6127 \bool_lazy_or:nnT
6128 { \str_if_eq_p:ee \l_tmpa_tl { * } }
6129 { \tl_if_blank_p:o \l_tmpa_tl }
6130 { \def \l_tmpa_tl { 1 } }
6131 \bool_lazy_or:nnT
6132 { \str_if_eq_p:ee \l_tmpb_tl { * } }
6133 { \tl_if_blank_p:o \l_tmpb_tl }
6134 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6135 \int_compare:nNnT \l_tmpb_tl > { #2 }
6136 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6137 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6138 { \clist_put_right:Nn #1 { #####1 } }
6139 }
6140 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6141 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6142 {
6143 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6144 {

```

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`).

```

6145 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6146 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6147 }
6148 \ignorespaces
6149 }

6150 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6151 { \@@_error:n { cellcolor~in~Block } }
6152 % \end{macrocode}
6153 %
6154 % \begin{macrocode}
6155 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6156 { \@@_error:n { rowcolor~in~Block } }
6157 % \end{macrocode}
6158 %
6159 % \bigskip
6160 % The following command will be linked to |\rowcolor| in the tabular.
6161 % \begin{macrocode}
6162 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6163 {
6164 \tl_gput_right:Ne \g_@@_pre_code_before_tl
6165 {
6166 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6167 { \int_use:N \c@iRow - \int_use:N \c@jCol }
6168 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6169 }
6170 \ignorespaces
6171 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by currying).

```

6172 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6173 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6174 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6175 {

```

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`.

```

6176 \seq_gclear:N \g_tmpa_seq
6177 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6178 { \@@_rowlistcolors_tabular:nnnn #1 }
6179 \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`.

```

6180 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6181 {
6182   { \int_use:N \c@iRow }
6183   { \exp_not:n { #1 } }
6184   { \exp_not:n { #2 } }
6185   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6186 }
6187 \ignorespaces
6188 }

```

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`).

```

6189 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6190 {
6191   \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).

```

6192   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6193   {
6194     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6195     {
6196       \@@_rowlistcolors
6197       [ \exp_not:n { #2 } ]
6198       { #1 - \int_eval:n { \c@iRow - 1 } }
6199       { \exp_not:n { #3 } }
6200       [ \exp_not:n { #4 } ]
6201     }
6202   }
6203 }

```

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.

```

6204 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:

```

```

6205 {
6206   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6207   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6208   \seq_gclear:N \g_@@_rowlistcolors_seq
6209 }

6210 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6211 {
6212   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6213   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6214 }

```

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.

```

6215 \NewDocumentCommand \@@_columnncolor_preamble { 0 { } m }
6216 {

```

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").

```

6217   \seq_if_in:NVF \g_@@_columnncolor_seq \c@jCol
6218   {
6219     \seq_gput_right:NV \g_@@_columnncolor_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).

```

6220     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6221     {
6222       \exp_not:N \columnncolor [ #1 ]
6223       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6224     }
6225   }
6226 }

```

```

6227 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6228 {
6229   \clist_map_inline:nn { #1 }
6230   {
6231     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6232     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6233     \columnncolor { nocolor } { ##1 }
6234   }
6235 }

6236 \cs_new_protected:Npn \@@_EmptyRow:n #1
6237 {
6238   \clist_map_inline:nn { #1 }
6239   {
6240     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6241     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6242     \rowcolor { nocolor } { ##1 }
6243   }
6244 }

```


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`.

```
6245 \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`.

```
6246 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6247 {
6248   \int_if_zero:nTF \l_@@_first_col_int
6249   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6250   {
6251     \int_if_zero:nTF \c@jCol
6252     {
6253       \int_compare:nNnF \c@iRow = { -1 }
6254       {
6255         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6256         { #1 }
6257       }
6258     }
6259     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6260   }
6261 }
```

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.

```
6262 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6263 {
6264   \int_if_zero:nF \c@iRow
6265   {
6266     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6267     {
6268       \int_compare:nNnT \c@jCol > \c_zero_int
6269       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6270     }
6271   }
6272 }
```

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`.

```
6273 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6274 {
6275   \IfPackageLoadedTF { tikz }
6276   {
6277     \IfPackageLoadedTF { booktabs }
6278     { #2 }
6279     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6280   }
```

```

6281 { \@@_error:nn { TopRule~without~tikz } { #1 } }
6282 }
6283 \NewExpandableDocumentCommand { \@@_TopRule } { }
6284 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6285 \cs_new:Npn \@@_TopRule_i:
6286 {
6287   \noalign \bgroup
6288     \peek_meaning:NTF [
6289       { \@@_TopRule_ii: }
6290       { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6291     }
6292 \NewDocumentCommand \@@_TopRule_ii: { o }
6293 {
6294   \tl_gput_right:Ne \g_@@_rules_tl
6295   {
6296     \@@_draw_hrule:n
6297     {
6298       position = \int_eval:n { \c@iRow + 1 } ,
6299       tikz =
6300       {
6301         line~width = #1 ,
6302         yshift = 0.25 \arrayrulewidth ,
6303         shorten~< = - 0.5 \arrayrulewidth
6304       } ,
6305       total~width = #1
6306     }
6307   }
6308   \skip_vertical:n { \belowrulesep + #1 }
6309   \egroup
6310 }
6311 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6312 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6313 \cs_new:Npn \@@_BottomRule_i:
6314 {
6315   \noalign \bgroup
6316     \peek_meaning:NTF [
6317       { \@@_BottomRule_ii: }
6318       { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6319     }
6320 \NewDocumentCommand \@@_BottomRule_ii: { o }
6321 {
6322   \tl_gput_right:Ne \g_@@_rules_tl
6323   {
6324     \@@_draw_hrule:n
6325     {
6326       position = \int_eval:n { \c@iRow + 1 } ,
6327       tikz =
6328       {
6329         line~width = #1 ,
6330         yshift = 0.25 \arrayrulewidth ,
6331         shorten~< = - 0.5 \arrayrulewidth
6332       } ,
6333       total~width = #1 ,
6334     }
6335   }
6336   \skip_vertical:N \aboverulesep
6337   \@@_create_row_node_i:
6338   \skip_vertical:n { #1 }
6339   \egroup
6340 }
6341 \NewExpandableDocumentCommand { \@@_MidRule } { }

```

```

6342 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6343 \cs_new:Npn \@@_MidRule_i:
6344 {
6345   \noalign \bgroup
6346   \peek_meaning:NTF [
6347     { \@@_MidRule_ii: }
6348     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6349   }
6350 \NewDocumentCommand \@@_MidRule_ii: { o }
6351 {
6352   \skip_vertical:N \aboverulesep
6353   \@@_create_row_node_i:
6354   \tl_gput_right:Ne \g_@@_rules_tl
6355   {
6356     \@@_draw_hrulerule:n
6357     {
6358       position = \int_eval:n { \c@iRow + 1 } ,
6359       tikz =
6360       {
6361         line-width = #1 ,
6362         yshift = 0.25 \arrayrulewidth ,
6363         shorten-< = - 0.5 \arrayrulewidth
6364       } ,
6365       total-width = #1 ,
6366     }
6367   }
6368   \skip_vertical:n { \belowrulesep + #1 }
6369   \egroup
6370 }

```

General system for drawing rules

```

6371 \AtBeginDocument
6372 {
6373   \cs_new_protected:Npe \@@_draw_rules:
6374   {
6375     \c_@@_pgfortikzpicture_tl
6376     \@@_draw_rules_i:
6377     \c_@@_endpgfortikzpicture_tl
6378   }
6379 }
6380 \cs_new_protected:Npn \@@_draw_rules_i:
6381 {
6382   \bool_lazy_all:nT
6383   {
6384     { \clist_if_empty_p:N \l_@@_corners_clist }
6385     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6386     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6387     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6388     { ! \l_@@_fix_vertex_bool }
6389   }
6390   {
6391     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6392     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6393   }
6394   \pgfrememberpicturepositiononpagetrue
6395   \pgf@relevantforpicturesizefalse
6396   \pgfsetrectcap
6397   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6398   \CT@arc@
6399   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:

```

```

6400 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6401 \g_@@_rules_tl
6402 }

```

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”.

```

6403 \keys_define:nn { nicematrix / rules-after }
6404 {
6405   position .int_set:N = \l_@@_position_int ,
6406   start .int_set:N = \l_@@_start_int ,
6407   start .initial:n = 1 ,
6408   end .int_set:N = \l_@@_end_int ,
6409   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6410   dotted .code:n =
6411     \bool_set_true:N \l_@@_dotted_bool
6412     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6413     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6414   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]`.

```

6415   color .code:n =
6416     \@@_set_CTarc:n { #1 }
6417     \CTarc@
6418     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6419   color .value_required:n = true ,
6420   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6421   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
  }
}

6422 total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6423 unknown .code:n =
6424   \@@_unknown_key:nn
6425     { nicematrix / rules-after }
6426     { Unknown-key-for-a-rule } ,
6427 tikz .value_required:n = true
6428 }

```

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.

```

6429 \AtBeginDocument
6430 {
6431   \IfPackageLoadedTF { tikz }
6432   {
6433     \keys_define:nn { nicematrix / rules-after }
6434     {
6435       tikz .code:n =
6436         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6437         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }

```

```

6438         \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6439         dashed .meta:n =
6440         {
6441             tikz = nicematrix / dashed ,
6442             total-width = \pgflinewidth
6443         }
6444     }
6445 }
6446 {
6447     \keys_define:nn { nicematrix / rules-after }
6448     { tikz .code:n = \@@_error:n { tikz-without~tikz } }
6449 }
6450 }

6451 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6452 {
6453     \int_set:Nn \l_@@_multiplicity_int { #1 }
6454     \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6455     \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6456 }

6457 \AtBeginDocument
6458 {
6459     \IfPackageLoadedT { tikz }
6460     {
6461         \tikzset
6462         {
6463             nicematrix / dashed / .style =
6464             {
6465                 dash-pattern = on~4~pt~off~2pt ,
6466                 line-cap = butt
6467             }
6468         }
6469         \cs_if_exist:cT { tikz@library@decorations@loaded }
6470         { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6471     }
6472 }

```

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.

```

6473 \cs_new_protected:Npn \@@_draw_vrule:n #1
6474 {

```

The group is for the options.

```

6475     {
6476         \int_set_eq:NN \l_@@_end_int \c@iRow
6477         \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).

```

6478         \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6479         \socket_use:n { nicematrix / draw-vrule }
6480     }
6481 }

```

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.

```
6482 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6483 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6484 {
6485   \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`.

```
6486   \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).

```
6487   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6488   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6489   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6490   \l_tmpa_tl
6491   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to true if we have to draw that rule.

```
6492     \@@_test_v:
6493     \bool_if:NTF \g_tmpa_bool
6494     {
6495       \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.

```
6496       { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6497       { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6498     }
6499     {
6500       \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6501       {
6502         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
```

The socket `vsegment` is defined below with its four plugs.

```
6503         \socket_use:n { nicematrix / vsegment }
6504         \int_zero:N \l_@@_segment_start_int
6505       }
6506     }
6507   }
6508   \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6509   {
6510     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6511     \socket_use:n { nicematrix / vsegment }
6512   }
6513   \group_end:
6514 }

6515 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6516 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6517 {
6518   \group_begin:
6519   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6520   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6521   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6522   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
```

```

6523     \socket_use:n { nicematrix / vsegment }
6524     \group_end:
6525 }

```

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.

```

6526 \cs_new_protected:Npn \@@_test_v:
6527 {
6528   \bool_gset_true:N \g_tmpa_bool
6529   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6530   \bool_if:NT \g_tmpa_bool
6531   {
6532     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6533     { \@@_test_vline_in_block:nnnnn ##1 }
6534     \bool_if:NT \g_tmpa_bool
6535     {
6536       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6537       { \@@_test_vline_in_block:nnnnn ##1 }
6538       \bool_if:NT \g_tmpa_bool
6539       {
6540         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6541         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6542       }
6543     }
6544   }
6545 }

6546 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6547 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6548 {
6549   {
6550     \int_compare:nNnTF \l_@@_position_int > \c@jCol
6551     { \bool_set_false:N \l_tmpa_bool }
6552     {
6553       \@@_test_h:
6554       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6555     }
6556     \int_compare:nNnTF \l_@@_position_int = 1
6557     { \bool_gset_false:N \g_tmpa_bool }
6558     {
6559       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6560       \@@_test_h:
6561     }
6562     \bool_gset:Nn \g_tmpa_bool
6563     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6564   }
6565   \bool_if:NT \g_tmpa_bool
6566   {
6567     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6568     \socket_use:n { nicematrix / vsegment }
6569     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6570   }
6571 }

6572 \cs_new_protected:Npn \@@_test_in_corner_v:
6573 {
6574   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6575   {
6576     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6577     { \bool_set_false:N \g_tmpa_bool }
6578   }

```

```

6579 {
6580   \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6581   {
6582     \int_compare:nNnTF \l_tmpb_tl = 1
6583     { \bool_set_false:N \g_tmpa_bool }
6584     {
6585       \@@_if_in_corner:nT
6586       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6587       { \bool_set_false:N \g_tmpa_bool }
6588     }
6589   }
6590 }
6591 }

```

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`.

```

6592 \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`.

```

6593 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6594 {
6595   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6596   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6597   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6598   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6599   \pgfusepathqstroke
6600 }
6601 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6602 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6603 {
6604   \dim_add:Nn \l_@@_x_initial_dim
6605   {
6606     - 0.5 \l_@@_rule_width_after_dim
6607     +
6608     ( \arrayrulewidth * \l_@@_multiplicity_int
6609       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6610   }
6611   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6612   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6613   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6614   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6615   \bool_lazy_and:nnT
6616   { \cs_if_exist_p:N \CT@drsc@ }
6617   { ! \tl_if_blank_p:o \CT@drsc@ }
6618   {
6619     {
6620       \CT@drsc@
6621       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6622       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6623       \dim_set:Nn \l_@@_tmpd_dim
6624       {
6625         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6626         * ( \l_@@_multiplicity_int - 1 )

```



```

6627     }
6628     \pgfpathrectanglecorners
6629     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6630     { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6631     \pgfusepath { fill }
6632   }
6633 }
6634 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6635 \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6636 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6637 {
6638   \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6639   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6640   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6641 }
6642 \pgfusepathqstroke
6643 }

6644 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6645 {
6646   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6647   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6648   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6649   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6650   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6651   \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`

```

6652   \@@_draw_line:
6653 }

6654 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6655 {
6656   {

```

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).

```

6657   \tl_if_empty:NF \l_@@_rule_color_tl
6658   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6659   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6660   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6661   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6662   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6663   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6664   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6665   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6666   ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6667 }
6668 }

```

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).

```

6669 \cs_new_protected:Npn \@@_draw_key_vlines:
6670 {
6671   {
6672     \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6673     \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currying in the following code.

```

6674     \str_if_eq:eeTF \l_@@_vlines_clist { all }
6675     { \@@_lines_step_inline:n \c@jCol }
6676     { \clist_map_inline:Nn \l_@@_vlines_clist }
6677     {
6678       \int_set:Nn \l_@@_position_int { ##1 }
6679       \socket_use:n { nicematrix / draw-vrule }
6680     }
6681   }
6682 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6683 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6684 {
6685   \int_step_inline:nnn
6686   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6687   {
6688     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6689     { #1 }
6690     { \int_eval:n { #1 + 1 } }
6691   }
6692 }

```

The horizontal rules

`\@@_draw_hrrule: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.

```

6693 \cs_new_protected:Npn \@@_draw_hrrule:n #1
6694 {
6695   {
6696     \int_set_eq:NN \l_@@_end_int \c@jCol
6697     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6698     \socket_use:nn { nicematrix / draw-hrule }
6699   }
6700 }

```

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.

```

6701 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6702 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6703 {
6704   \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`.

```

6705   \tl_set:N \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).

```

6706   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6707   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6708   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6709   \l_tmpb_tl
6710   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6711     \@@_test_h:
6712     \bool_if:NTF \g_tmpa_bool
6713     {
6714       \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.

```

6715         { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6716         { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6717     }
6718     {
6719       \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6720       {
6721         \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

6722         \socket_use:n { nicematrix / hsegment }
6723         \int_zero:N \l_@@_segment_start_int
6724     }
6725   }
6726 }
6727 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6728 {
6729   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6730   \socket_use:n { nicematrix / hsegment }
6731 }
6732 \group_end:
6733 }

6734 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6735 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6736 {
6737   \group_begin:
6738   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6739   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6740   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6741   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6742   \socket_use:n { nicematrix / hsegment }
6743   \group_end:
6744 }

```

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.

```

6745 \cs_new_protected:Npn \@@_test_h:
6746 {

```

```

6747 \bool_gset_true:N \g_tmpa_bool
6748 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6749 \bool_if:NT \g_tmpa_bool
6750 {
6751   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6752   { \@@_test_hline_in_block:nnnnn ##1 }
6753   \bool_if:NT \g_tmpa_bool
6754   {
6755     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6756     { \@@_test_hline_in_block:nnnnn ##1 }
6757     \bool_if:NT \g_tmpa_bool
6758     {
6759       \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6760       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6761     }
6762   }
6763 }
6764 }

6765 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6766 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6767 {
6768   {
6769     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6770     { \bool_set_false:N \l_tmpa_bool }
6771     {
6772       \@@_test_v:
6773       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6774     }
6775     \int_compare:nNnTF \l_@@_position_int = 1
6776     { \bool_gset_false:N \g_tmpa_bool }
6777     {
6778       \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6779       \@@_test_v:
6780     }
6781     \bool_gset:Nn \g_tmpa_bool
6782     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6783   }
6784   \bool_if:NT \g_tmpa_bool
6785   {
6786     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6787     \socket_use:n { nicematrix / hsegment }
6788     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6789   }
6790 }

6791 \cs_new_protected:Npn \@@_test_in_corner_h:
6792 {
6793   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6794   {
6795     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6796     { \bool_set_false:N \g_tmpa_bool }
6797   }
6798   {
6799     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6800     {
6801       \int_compare:nNnTF \l_tmpa_tl = 1
6802       { \bool_set_false:N \g_tmpa_bool }
6803       {
6804         \@@_if_in_corner:nT
6805         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6806         { \bool_set_false:N \g_tmpa_bool }

```

```

6807         }
6808     }
6809 }
6810 }

```

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`.

```

6811 \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`.

```

6812 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6813 {
6814     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6815     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6816     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6817     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6818     \pgfusepathqstroke
6819 }
6820 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
6821 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6822 {
6823     \dim_add:Nn \l_@@_y_initial_dim
6824     {
6825         - 0.5 \l_@@_rule_width_after_dim
6826         +
6827         ( \arrayrulewidth * \l_@@_multiplicity_int
6828           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6829     }
6830     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6831     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6832     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6833     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6834     \bool_lazy_and:nnT
6835     { \cs_if_exist_p:N \CT@drsc@ }
6836     { ! \tl_if_blank_p:o \CT@drsc@ }
6837     {
6838         {
6839             \CT@drsc@
6840             \dim_set:Nn \l_@@_tmpd_dim
6841             {
6842                 \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6843                 * ( \l_@@_multiplicity_int - 1 )
6844             }
6845             \pgfpathrectanglecorners
6846             { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6847             { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6848             \pgfusepathqfill
6849         }
6850     }
6851     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6852     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6853     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6854     {

```

```

6855 \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6856 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6857 \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6858 }
6859 \pgfusepathqstroke
6860 }

```

Now, the case of a dotted line.

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

```

6861 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
6862 {
6863 \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6864 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6865 \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6866 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6867 \int_compare:nNnT \l_@@_segment_start_int = 1
6868 {
6869 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6870 \bool_if:NF \g_@@_delims_bool
6871 { \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.

```

6872 \tl_if_eq:NnF \g_@@_left_delim_tl (
6873 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6874 )
6875 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6876 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6877 \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6878 {
6879 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6880 \bool_if:NF \g_@@_delims_bool
6881 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6882 \tl_if_eq:NnF \g_@@_right_delim_tl )
6883 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6884 }

```

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`

```

6885 \@@_draw_line:
6886 }

```

```

6887 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6888 {
6889 {

```

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).

```

6890 \tl_if_empty:NF \l_@@_rule_color_tl
6891 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6892 \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6893 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6894 \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6895 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6896 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6897 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6898 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6899 -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6900 }
6901 }

```

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).

```

6902 \cs_new_protected:Npn \@@_draw_key_hlines:
6903 {
6904 {
6905 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6906 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6907 \str_if_eq:eeTF \l_@@_hlines_clist { all }
6908 { \@@_lines_step_inline:n \c@iRow }
6909 { \clist_map_inline:Nn \l_@@_hlines_clist {
6910 {
6911 \int_set:Nn \l_@@_position_int { ##1 }
6912 \socket_use:n { nicematrix / draw-hrule }
6913 }
6914 }
6915 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6916 \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.

```

6917 \cs_set:Npn \@@_Hline_i:n #1
6918 {
6919 \peek_remove_spaces:n
6920 {
6921 \peek_meaning:NTF \Hline
6922 { \@@_Hline_ii:nn { #1 + 1 } }
6923 { \@@_Hline_iii:n { #1 } }
6924 }
6925 }
6926 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6927 \cs_set:Npn \@@_Hline_iii:n #1
6928 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

```

```

6929 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6930 {
6931   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6932   \skip_vertical:N \l_@@_rule_width_before_dim
6933   \tl_gput_right:Ne \g_@@_rules_tl
6934   {

```

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:

```

6935   {
6936     \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6937     \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6938     \dim_set:Nn \l_@@_rule_width_after_dim
6939       { \dim_use:N \l_@@_rule_width_before_dim }
6940     \@@_draw_hrule:n { #2 }
6941   }
6942 }
6943 \egroup
6944 }

```

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`).

```

6945 \cs_new_protected:Npn \@@_custom_line:n #1
6946 {
6947   \str_clear:N \l_@@_letter_str
6948   \str_clear:N \l_@@_command_str
6949   \str_clear:N \l_@@_ccommand_str
6950   \tl_clear_new:N \l_@@_other_keys_tl
6951   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6952   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6953   {
6954     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6955     \str_set:Ne \l_@@_command_str
6956       { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6957   }
6958   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6959   {
6960     \str_set:Ne \l_@@_ccommand_str
6961       { \str_tail:N \l_@@_ccommand_str }
6962     \str_set:Ne \l_@@_ccommand_str
6963       { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6964   }

```

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.


```

6965 \bool_lazy_all:nTF
6966 {
6967   { \str_if_empty_p:N \l_@@_letter_str }
6968   { \str_if_empty_p:N \l_@@_command_str }
6969   { \str_if_empty_p:N \l_@@_ccommand_str }
6970 }
6971 { \@@_error:n { No~letter~and~no~command } }
6972 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6973 }
6974 \keys_define:nn { nicematrix / custom-line }
6975 {
6976   letter .str_set:N = \l_@@_letter_str ,
6977   letter .value_required:n = true ,
6978   command .str_set:N = \l_@@_command_str ,
6979   command .value_required:n = true ,
6980   ccommand .str_set:N = \l_@@_ccommand_str ,
6981   ccommand .value_required:n = true
6982 }

```

```

6983 \cs_new_protected:Npn \@@_custom_line_i:n #1
6984 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6985 \bool_set_false:N \l_@@_tikz_rule_bool
6986 \bool_set_false:N \l_@@_dotted_rule_bool
6987 \bool_set_false:N \l_@@_color_bool
6988 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6989 \bool_if:NT \l_@@_tikz_rule_bool
6990 {
6991   \IfPackageLoadedF { tikz }
6992   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6993   \bool_if:NT \l_@@_color_bool
6994   { \@@_error:n { color~in~custom~line~with~tikz } }
6995 }
6996 \bool_if:NT \l_@@_dotted_rule_bool
6997 {
6998   \int_compare:nNnT \l_@@_multiplicity_int > 1
6999   { \@@_error:n { key~multiplicity~with~dotted } }
7000 }
7001 \str_if_empty:NF \l_@@_letter_str
7002 {
7003   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7004   { \@@_error:n { Several~letters } }
7005   {
7006     \tl_if_in:NoTF
7007     \c_@@_forbidden_letters_str
7008     \l_@@_letter_str
7009     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
7010   }

```

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.

```

7011 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7012 { \@@_v_custom_line:nn { #1 } }
7013 }
7014 }
7015 }
7016 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7017 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7018 }
7019 \cs_generate_variant:Nn \@@_custom_line_i:n { o }

```

```

7020 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
7021 \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.

```

7022 \keys_define:nn { nicematrix / custom-line-bis }
7023 {
7024   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7025   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7026   color .value_required:n = true ,
7027   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7028   tikz .value_required:n = true ,
7029   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7030   dotted .value_forbidden:n = true ,
7031   dashed .meta:n =
7032   {
7033     tikz = nicematrix / dashed ,
7034     total-width = \pgflinewidth
7035   } ,
7036   total-width .code:n = { } ,
7037   total-width .value_required:n = true ,
7038   sep-color .code:n = { } ,
7039   sep-color .value_required:n = true ,
7040   unknown .code:n =
7041     \@@_unknown_key:nn
7042     { nicematrix / custom-line-bis }
7043     { Unknown-key-for~custom-line }
7044 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7045 \bool_new:N \l_@@_dotted_rule_bool
7046 \bool_new:N \l_@@_tikz_rule_bool
7047 \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).

```

7048 \keys_define:nn { nicematrix / custom-line-width }
7049 {
7050   multiplicity .int_set:N = \l_@@_tmpc_int ,
7051   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7052   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7053     \bool_set_true:N \l_@@_total_width_bool ,
7054   total-width .value_required:n = true ,
7055   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7056 }

```

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_hrline:n` (which is in the internal `\CodeAfter`).

```

7057 \cs_new_protected:Npn \@@_h_custom_line:n #1
7058 {

```

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`).

```

7059   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7060   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7061 }

```

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`).

```
7062 \cs_new_protected:Npn \@@_c_custom_line:n #1
7063 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
7064   \exp_args:Nc \DeclareExpandableDocumentCommand
7065     { nicematrix - \l_@@_ccommand_str }
7066     { 0 { } m }
7067     {
7068       \noalign
7069       {
7070         \@@_compute_rule_width:n { #1 , ##1 }
7071         \skip_vertical:n \l_@@_rule_width_before_dim
7072         \clist_map_inline:nn
7073           { ##2 }
7074           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7075       }
7076     }
7077   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7078 }
```

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*.

```
7079 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7080 {
7081   \tl_if_in:nnTF { #2 } { - }
7082     { \@@_cut_on_hyphen:w #2 \q_stop }
7083     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7084   \tl_gput_right:Ne \g_@@_rules_tl
7085   {
7086     \@@_draw_hrule:n
7087     {
7088       #1 ,
7089       start = \l_tmpa_tl ,
7090       end = \l_tmpb_tl ,
7091       position = \int_eval:n { \c@iRow + 1 } ,
7092       total-width = \dim_use:N \l_@@_rule_width_before_dim
7093     }
7094   }
7095 }

7096 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7097 {
7098   \bool_set_false:N \l_@@_tikz_rule_bool
7099   \bool_set_false:N \l_@@_total_width_bool
7100   \bool_set_false:N \l_@@_dotted_rule_bool
7101   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7102   \bool_if:NF \l_@@_total_width_bool
7103   {
7104     \bool_if:NTF \l_@@_dotted_rule_bool
7105       { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7106       {
7107         \bool_if:NF \l_@@_tikz_rule_bool
7108         {
7109           \dim_set:Nn \l_@@_rule_width_before_dim
7110             {
7111               \arrayrulewidth * \l_@@_tmpc_int
7112               + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7113             }
7114         }
7115       }
7116   }
7117 }
```

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

```

7118 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7119 {
7120   \str_if_eq:nnTF { #2 } { [ ] }
7121   { \@@_v_custom_line_i:nw { #1 } [ ] }
7122   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7123 }
7124 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7125 { \@@_v_custom_line:nn { #1 , #2 } }
7126 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7127 {
7128   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7129 \tl_gput_right:Ne \g_@@_array_preamble_tl
7130 {
7131   \exp_not:N !
7132   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7133 }
7134 \tl_gput_right:Ne \g_@@_rules_tl
7135 {

```

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.

```

7136 {
7137   \dim_set:Nn \l_@@_rule_width_after_dim
7138   { \dim_use:N \l_@@_rule_width_before_dim }
7139   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7140   \@@_draw_vrule:n { #2 }
7141 }
7142 }
7143 \@@_rec_preamble:n #1
7144 }
7145 \@@_custom_line:n
7146 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

```

The key default-line

```

7147 \keys_define:nn { nicematrix / default-line }
7148 {
7149   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7150   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7151   color .value_required:n = true ,
7152   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7153   dotted .value_forbidden:n = true ,
7154   total-width .code:n = { } ,
7155   total-width .value_required:n = true ,
7156   sep-color .code:n = { } ,
7157   sep-color .value_required:n = true ,
7158   unknown .code:n =
7159     \@@_unknown_key:nn
7160     { nicematrix / default-line }
7161     { Unknown-key~for~default-line }
7162 }

```

```

7163 \AtBeginDocument
7164 {
7165   \IfPackageLoadedT { tikz }
7166   {
7167     \keys_define:nn { nicematrix / default-line }
7168     {
7169       tikz .code:n =
7170         \bool_set_true:N \l_@@_tikz_rule_bool
7171         \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7172       tikz .value_required:n = true ,
7173       dashed .meta:n =
7174       {
7175         tikz = nicematrix / dashed ,
7176         total-width = \pgflinewidth
7177       }
7178     }
7179     \@@_custom_line:n
7180     {
7181       letter = ; ,
7182       command = \hdashedline ,
7183       ccommand = \cdashedline ,
7184       tikz = nicematrix / dashed ,
7185       total-width = \pgflinewidth
7186     }
7187   }
7188 }
7189 \cs_new_protected:Npn \@@_default_line:n #1
7190 {
7191   \bool_set_false:N \l_@@_tikz_rule_bool
7192   \bool_set_false:N \l_@@_dotted_rule_bool
7193   \bool_set_false:N \l_@@_color_bool
7194   \keys_set:nn { nicematrix / default-line } { #1 }
7195   \bool_if:NT \l_@@_tikz_rule_bool
7196   {
7197     \IfPackageLoadedF { tikz }
7198     { \@@_error:n { tikz~in~custom~line~without~tikz } }
7199     \bool_if:NT \l_@@_color_bool
7200     { \@@_error:n { color~in~custom~line~with~tikz } }
7201   }
7202   \bool_if:NT \l_@@_dotted_rule_bool
7203   {
7204     \int_compare:nNnT \l_@@_multiplicity_int > 1
7205     { \@@_error:n { key~multiplicity~with~dotted } }
7206   }
7207   \bool_if:NTF \l_@@_tikz_rule_bool
7208   {
7209     \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7210     \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7211   }
7212   {
7213     \bool_if:NTF \l_@@_dotted_rule_bool
7214     {
7215       \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7216       \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7217     }
7218     {
7219       \bool_if:NTF \l_@@_multiple_rule_bool
7220       {
7221         \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7222         \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7223       }
7224     }
7225   }

```

```
7226 }
```

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.

```
7227 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7228 {
7229   \int_compare:nNnT \l_tmpa_tl > { #1 }
7230   {
7231     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7232     {
7233       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7234       {
7235         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7236         { \bool_gset_false:N \g_tmpa_bool }
7237       }
7238     }
7239   }
7240 }
```

The same for vertical rules.

```
7241 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7242 {
7243   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7244   {
7245     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7246     {
7247       \int_compare:nNnT \l_tmpb_tl > { #2 }
7248       {
7249         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7250         { \bool_gset_false:N \g_tmpa_bool }
7251       }
7252     }
7253   }
7254 }

7255 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7256 {
7257   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7258   {
7259     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7260     {
7261       \int_compare:nNnTF \l_tmpa_tl = { #1 }
7262       { \bool_gset_false:N \g_tmpa_bool }
7263       {
7264         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7265         { \bool_gset_false:N \g_tmpa_bool }
7266       }
7267     }
7268   }
7269 }

7270 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7271 {
7272   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7273   {
7274     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7275     {
7276       \int_compare:nNnTF \l_tmpb_tl = { #2 }
7277       { \bool_gset_false:N \g_tmpa_bool }
7278       {
7279         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
```

```

7280         { \bool_gset_false:N \g_tmpa_bool }
7281     }
7282 }
7283 }
7284 }

```

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.

```

7285 \cs_new_protected:Npn \@@_compute_corners:
7286 {
7287     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7288     { \@@_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`).

```

7289     \clist_clear:N \l_@@_corners_cells_clist
7290     \clist_map_inline:Nn \l_@@_corners_cells_clist
7291     {
7292         \str_case:nnF { ##1 }
7293         {
7294             { NW }
7295             { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7296             { NE }
7297             { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7298             { SW }
7299             { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7300             { SE }
7301             { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7302             { N }
7303             { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7304             { S }
7305             { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7306             { E }
7307             { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7308             { W }
7309             { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7310         }
7311         { \@@_error:nn { bad~corner } { ##1 } }
7312     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7313     \clist_if_empty:NF \l_@@_corners_cells_clist
7314     {

```

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.

```

7315     \tl_gput_right:Ne \g_@@_aux_tl
7316     {
7317         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7318         { \l_@@_corners_cells_clist }
7319     }
7320 }
7321 }

```

```

7322 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7323 {
7324   \int_step_inline:nnn { #1 } { #3 }
7325   {
7326     \int_step_inline:nnn { #2 } { #4 }
7327     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
7328   }
7329 }

7330 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7331 {
7332   \cs_if_exist:cTF
7333   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7334   \prg_return_true:
7335   \prg_return_false:
7336 }

```

“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.

```

7337 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7338 {

```

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.

```

7339   \bool_set_false:N \l_tmpa_bool
7340   \int_zero_new:N \l_@@_last_empty_row_int
7341   \int_set:Nn \l_@@_last_empty_row_int { #1 }
7342   \int_step_inline:nnnn { #1 } { #3 } { #5 }
7343   {
7344     \bool_lazy_or:nnTF
7345     {
7346       \cs_if_exist_p:c
7347       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7348     }
7349     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7350     { \bool_set_true:N \l_tmpa_bool }
7351   }
7352   \bool_if:NF \l_tmpa_bool
7353   { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7354 }
7355 }

```

Now, you determine the last empty cell in the row of number 1.

```

7356   \bool_set_false:N \l_tmpa_bool
7357   \int_zero_new:N \l_@@_last_empty_column_int
7358   \int_set:Nn \l_@@_last_empty_column_int { #2 }
7359   \int_step_inline:nnnn { #2 } { #4 } { #6 }
7360   {
7361     \bool_lazy_or:nnTF
7362     {

```



```

7363     \cs_if_exist_p:c
7364     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7365   }
7366   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7367   { \bool_set_true:N \l_tmpa_bool }
7368   {
7369     \bool_if:NF \l_tmpa_bool
7370     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7371   }
7372 }

```

Now, we loop over the rows.

```

7373   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7374   {

```

We treat the row number ##1 with another loop.

```

7375     \bool_set_false:N \l_tmpa_bool
7376     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7377     {
7378       \bool_lazy_or:nnTF
7379       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7380       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7381       { \bool_set_true:N \l_tmpa_bool }
7382       {
7383         \bool_if:NF \l_tmpa_bool
7384         {
7385           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7386           \clist_put_right:Nn
7387             \l_@@_corners_cells_clist
7388             { ##1 - #####1 }
7389           \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7390         }
7391       }
7392     }
7393   }
7394 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7395 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7396 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

```

7397 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7398 {
7399   \int_step_inline:nn { #1 }
7400   {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7401     \bool_set_false:N \l_tmpa_bool
7402     \int_step_inline:nnnn { #2 } { #3 } { #4 }
7403     {
7404       \bool_lazy_or:nnTF
7405       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 } }
7406       { \@@_if_in_block_p:nn { #####1 } { ##1 } }
7407       { \bool_set_true:N \l_tmpa_bool }
7408       {
7409         \bool_if:NF \l_tmpa_bool
7410         {
7411           \clist_put_right:Nn
7412             \l_@@_corners_cells_clist
7413             { #####1 - ##1 }
7414           \cs_set_nopar:cpn { @@ _ corner _ #####1 - ##1 } { }

```

```

7415     }
7416   }
7417 }
7418 }
7419 }

```

```

7420 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7421 {
7422   \int_step_inline:nn { #1 }
7423   {

```

We will raise the flag: `\l_tmpa_bool` when we will find an non-empty cell.

```

7424     \bool_set_false:N \l_tmpa_bool
7425     \int_step_inline:nnnn { #2 } { #3 } { #4 }
7426     {
7427       \bool_lazy_or:nnTF
7428       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7429       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7430       { \bool_set_true:N \l_tmpa_bool }
7431       {
7432         \bool_if:NF \l_tmpa_bool
7433         {
7434           \clist_put_right:Nn
7435           \l_@@_corners_cells_clist
7436           { ##1 - #####1 }
7437           \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7438         }
7439       }
7440     }
7441   }
7442 }

```

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.

```

7443 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7444 \keys_define:nn { nicematrix / NiceMatrixBlock }
7445 {
7446   auto-columns-width .code:n =
7447   {
7448     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7449     \dim_gzero_new:N \g_@@_max_cell_width_dim
7450     \bool_set_true:N \l_@@_auto_columns_width_bool
7451   }
7452 }

7453 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7454 {
7455   \int_gincr:N \g_@@_NiceMatrixBlock_int
7456   \dim_zero:N \l_@@_columns_width_dim
7457   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7458   \bool_if:NT \l_@@_block_auto_columns_width_bool
7459   {

```

```

7460 \cs_if_exist:cT
7461 { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7462 {
7463   \dim_set:Nn \l_@@_columns_width_dim
7464   {
7465     \use:c
7466     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7467   }
7468 }
7469 }
7470 }

```

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`).

```

7471 {
7472   \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.

```

7473 { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7474 {
7475   \bool_if:NT \l_@@_block_auto_columns_width_bool
7476   {
7477     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7478     \iow_shipout:Ne \@mainaux
7479     {
7480       \cs_gset:cpn
7481       { @@ _ 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.

```

7482       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7483     }
7484     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7485   }
7486 }
7487 \ignorespacesafterend
7488 }

```

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).

```

7489 \cs_new_protected:Npn \@@_create_extra_nodes:
7490 {
7491   \bool_if:nTF \l_@@_medium_nodes_bool
7492   {
7493     \bool_if:NTF \l_@@_no_cell_nodes_bool
7494     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7495     {
7496       \bool_if:NTF \l_@@_large_nodes_bool
7497       \@@_create_medium_and_large_nodes:
7498       \@@_create_medium_nodes:
7499     }
7500   }
7501   {
7502     \bool_if:NT \l_@@_large_nodes_bool
7503     {

```

```

7504         \bool_if:NTF \l_@@_no_cell_nodes_bool
7505         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7506         \@@_create_large_nodes:
7507     }
7508 }
7509 }

```

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`.

```

7510 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7511 {
7512     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7513     {
7514         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7515         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7516         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7517         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7518     }
7519     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7520     {
7521         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7522         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7523         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7524         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7525     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7526     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7527     {
7528         \int_step_variable:nnNn
7529         \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.

```

7530         {
7531             \cs_if_exist:cT
7532             { 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`.

```

7533         {
7534             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
7535             \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7536             { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7537             \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7538             {
7539                 \dim_set:cn { l_@@_column_ \@@_j: _min_dim }

```

```

7540         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7541     }

```

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`.

```

7542     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7543     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7544     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7545     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7546     {
7547         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7548         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
7549     }
7550 }
7551 }
7552 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7553 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7554 {
7555     \dim_compare:nNnT
7556     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7557     {
7558         \@@_qpoint:n { row - \@@_i: - base }
7559         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7560         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7561     }
7562 }
7563 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7564 {
7565     \dim_compare:nNnT
7566     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7567     {
7568         \@@_qpoint:n { col - \@@_j: }
7569         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7570         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7571     }
7572 }
7573 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7574 \cs_new_protected:Npn \@@_create_medium_nodes:
7575 {
7576     \pgfpicture
7577     \pgfrememberpicturepositiononpagetrue
7578     \pgf@relevantforpicturesizefalse
7579     \@@_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”.

```

7580     \tl_set:Nn \l_@@_suffix_tl { -medium }
7581     \@@_create_nodes:
7582     \endpgfpicture
7583 }

```

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

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7584 \cs_new_protected:Npn \@@_create_large_nodes:
7585 {
7586   \pgfpicture
7587     \pgfrememberpicturepositiononpagetrue
7588     \pgf@relevantforpicturesizefalse
7589     \@@_computations_for_medium_nodes:
7590     \@@_computations_for_large_nodes:
7591     \tl_set:Nn \l_@@_suffix_tl { - large }
7592     \@@_create_nodes:
7593   \endpgfpicture
7594 }

7595 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7596 {
7597   \pgfpicture
7598     \pgfrememberpicturepositiononpagetrue
7599     \pgf@relevantforpicturesizefalse
7600     \@@_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”.

```

7601   \tl_set:Nn \l_@@_suffix_tl { - medium }
7602   \@@_create_nodes:
7603   \@@_computations_for_large_nodes:
7604   \tl_set:Nn \l_@@_suffix_tl { - large }
7605   \@@_create_nodes:
7606 \endpgfpicture
7607 }

```

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.

```

7608 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7609 {
7610   \int_set:Nn \l_@@_first_row_int 1
7611   \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`.

```

7612   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7613   {
7614     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7615     {
7616       (
7617         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7618         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7619       )
7620       / 2
7621     }
7622     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7623     { l_@@_row _ \@@_i: _ min_dim }
7624   }
7625   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7626   {
7627     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7628     {
7629       (
7630         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7631         \dim_use:c
7632         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7633       )
7634       / 2
7635     }

```

```

7636     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7637     { l_@@_column _ \@@_j: _ max _ dim }
7638 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7639 \dim_sub:cn
7640 { l_@@_column _ 1 _ min _ dim }
7641 \l_@@_left_margin_dim
7642 \dim_add:cn
7643 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7644 \l_@@_right_margin_dim
7645 }

```

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).

```

7646 \cs_new_protected:Npn \@@_create_nodes:
7647 {
7648   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7649   {
7650     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7651     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7652       \@@_pgf_rect_node:nnnnn
7653       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7654       { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
7655       { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
7656       { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
7657       { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
7658       \str_if_empty:NF \l_@@_name_str
7659       {
7660         \pgfnodealias
7661         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7662         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7663       }
7664     }
7665   }
7666   \int_step_inline:nn \c@iRow
7667   {
7668     \pgfnodealias
7669     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7670     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7671   }
7672   \int_step_inline:nn \c@jCol
7673   {
7674     \pgfnodealias
7675     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7676     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7677   }
7678   \pgfnodealias
7679   { \@@_env: - last - last \l_@@_suffix_tl }
7680   { \@@_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 .

```

7681 \seq_map_pairwise_function:NNN
7682 \g_@@_multicolumn_cells_seq
7683 \g_@@_multicolumn_sizes_seq

```

```

7684 \@@_node_for_multicolumn:nn
7685 }

7686 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7687 {
7688   \cs_set_nopar:Npn \@@_i: { #1 }
7689   \cs_set_nopar:Npn \@@_j: { #2 }
7690 }

```

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”).

```

7691 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7692 {
7693   \@@_extract_coords_values: #1 \q_stop
7694   \@@_pgf_rect_node:nnnnn
7695   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7696   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7697   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7698   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7699   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7700   \str_if_empty:NF \l_@@_name_str
7701   {
7702     \pgfnodealias
7703     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7704     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7705   }
7706 }

```

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).

```

7707 \keys_define:nn { nicematrix / BlockFirstPass }
7708 {
7709   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7710           \bool_set_true:N \l_@@_p_block_bool ,
7711   j .value_forbidden:n = true ,
7712   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7713   l .value_forbidden:n = true ,
7714   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7715   r .value_forbidden:n = true ,
7716   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7717   c .value_forbidden:n = true ,
7718   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7719   L .value_forbidden:n = true ,
7720   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,
7721   R .value_forbidden:n = true ,
7722   C .code:n = \str_set:Nn \l_@@_hpos_block_str C ,
7723   C .value_forbidden:n = true ,
7724   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7725   t .value_forbidden:n = true ,
7726   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7727   T .value_forbidden:n = true ,

```



```

7728 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7729 b .value_forbidden:n = true ,
7730 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7731 B .value_forbidden:n = true ,
7732 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7733 m .value_forbidden:n = true ,
7734 v-center .meta:n = m ,
7735 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7736 p .value_forbidden:n = true ,
7737 respect-arraystretch .code:n =
7738   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7739 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.

```

7740 color .code:n =
7741   \@@_color:n { #1 }
7742   \tl_set_rescan:Nnn
7743     \l_@@_draw_tl
7744     { \char_set_catcode_other:N ! }
7745     { #1 } ,
7746 color .value_required:n = true ,
7747 }

```

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.

```

7748 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7749 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7750 {

```

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).

```

7751   \tl_if_blank:nTF { #2 }
7752     { \@@_Block_ii:nnnnn 1 1 }
7753     {
7754       \tl_if_in:nnTF { #2 } { - }
7755       {
7756         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7757         \@@_Block_i_czech:w \@@_Block_i:w
7758         #2 \q_stop
7759       }
7760       {
7761         \@@_error:nn { Bad~argument~for~Block } { #2 }
7762         \@@_Block_ii:nnnnn 1 1
7763       }
7764     }
7765   { #1 } { #3 } { #4 }
7766   \ignorespaces
7767 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7768 \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`.

```

7769 {
7770   \char_set_catcode_active:N -
7771   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7772 }

```

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.

```

7773 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7774 {

```

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).

```

7775 \bool_lazy_or:nnTF
7776 { \tl_if_blank_p:n { #1 } }
7777 { \str_if_eq_p:ee { * } { #1 } }
7778 { \int_set:Nn \l_tmpa_int { 100 } }
7779 { \int_set:Nn \l_tmpa_int { #1 } }
7780 \bool_lazy_or:nnTF
7781 { \tl_if_blank_p:n { #2 } }
7782 { \str_if_eq_p:ee { * } { #2 } }
7783 { \int_set:Nn \l_tmpb_int { 100 } }
7784 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7785 \int_compare:nNnTF \l_tmpb_int = 1
7786 {
7787   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7788   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7789   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7790 }
7791 { \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.

```

7792 \keys_set_known:nn { nicematrix / BlockFirstPass } { #3 }
7793 \tl_set:Ne \l_tmpa_tl
7794 {
7795   { \int_use:N \c@iRow }
7796   { \int_use:N \c@jCol }
7797   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7798   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7799 }

```

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).

```

7800 \bool_set_false:N \l_tmpa_bool
7801 \bool_if:NT \l_@@_amp_in_blocks_bool

```

\tl_if_in:nnT is slightly faster than \str_if_in:nnT.

```

7802 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7803 \bool_case:nF
7804 {
7805   \l_tmpa_bool { \@@_Block_vii:eennn }
7806   \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7807         \l_@@_X_bool                                { \@@_Block_v:eennn }
7808         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7809         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7810         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7811     }
7812     { \@@_Block_v:eennn }
7813     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7814 }

```

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.

```

7815 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7816 {
7817     \int_gincr:N \g_@@_block_box_int
7818     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7819     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7820     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7821     {
7822         \tl_gput_right:Ne \g_@@_rules_tl
7823         {
7824             \@@_draw_diagbox:nnnnnn
7825             { \int_use:N \c@iRow }
7826             { \int_use:N \c@jCol }
7827             { \int_eval:n { \c@iRow + #1 - 1 } }
7828             { \int_eval:n { \c@jCol + #2 - 1 } }
7829             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7830             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7831         }
7832     }
7833     \box_gclear_new:c
7834     { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

7835     \hbox_gset:cn
7836     { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7837     {

```

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).

```

7838     \tl_if_empty:NTF \l_@@_color_tl
7839     { \int_compare:nNnT { #2 } = 1 \set@color }
7840     { \@@_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`.

```

7841      \int_compare:nNnT { #1 } = 1
7842      {
7843          \int_if_zero:nTF \c@iRow
7844          {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7845      \cs_set_eq:NN \Block \@@_NullBlock:
7846      \l_@@_code_for_first_row_tl
7847      }
7848      {
7849          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7850          {
7851              \cs_set_eq:NN \Block \@@_NullBlock:
7852              \l_@@_code_for_last_row_tl
7853          }
7854      }
7855      \g_@@_row_style_tl
7856      }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7857      \@@_reset_arraystretch:
7858      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7859      #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.).

```

7860      \@@_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}`.

```

7861      \bool_if:nTF \l_@@_tabular_bool
7862      {
7863          \bool_lazy_all:nTF
7864          {
7865              { \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.

```
7866      { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7867      { ! \g_@@_rotate_bool }
7868    }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7869      {
7870      \use:e
7871      {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7872      \exp_not:N \begin { minipage }
7873      [ \str_lowercase:f \l_@@_vpos_block_str ]
7874      { \l_@@_col_width_dim }
7875      \str_case:on \l_@@_hpos_block_str
7876      { c \centering r \raggedleft l \raggedright }
7877    }
7878    #5
7879    \end { minipage }
7880  }
```

In the other cases, we use a `{tabular}`.

```
7881      {
7882      \use:e
7883      {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7884      \exp_not:N \begin { tabular }
7885      [ \str_lowercase:f \l_@@_vpos_block_str ]
7886      { @ { } \l_@@_hpos_block_str @ { } }
7887    }
7888    #5
7889    \end { tabular }
7890  }
7891}
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7892      {
7893      $ % $
7894      \bool_if:NT \l_@@_small_bool % 2026/04/05
7895      {
7896      \def \arraystretch { 0.47 }
7897      \dim_set:Nn \arraycolsep { 1.45 pt }
7898      }
7899      \use:e
7900      {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7901      \exp_not:N \begin { array }
7902      [ \str_lowercase:f \l_@@_vpos_block_str ]
7903      { @ { } \l_@@_hpos_block_str @ { } }
7904    }
7905    \@@_tuning_key_small: % 2026/04/05
7906    #5
7907    \end { array }
7908    $ % $
7909  }
7910}
```

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).

```
7911    \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.

```

7912     \int_compare:nNnT { #2 } = 1
7913     {
7914         \dim_gset:Nn \g_@@_blocks_wd_dim
7915         {
7916             \dim_max:nn
7917             \g_@@_blocks_wd_dim
7918             {
7919                 \box_wd:c
7920                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7921             }
7922         }
7923     }

```

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.

```

7924     \int_compare:nNnT { #1 } = 1
7925     {
7926         \bool_lazy_any:nT
7927         {
7928             { \str_if_empty_p:N \l_@@_vpos_block_str }
7929             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7930             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7931         }
7932         \@@_adjust_blocks_ht_dp:
7933     }
7934     \seq_gput_right:Ne \g_@@_blocks_seq
7935     {
7936         \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.

```

7937     {
7938         \exp_not:n { #3 } ,
7939         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7940         \bool_if:NT \g_@@_rotate_bool
7941         {
7942             \bool_if:NTF \g_@@_rotate_c_bool
7943             { m }
7944             {
7945                 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7946                 { T }
7947             }
7948         }
7949     }
7950     {
7951         \box_use_drop:c
7952         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7953     }
7954 }
7955 \bool_set_false:N \g_@@_rotate_c_bool
7956 }
7957 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7958 {
7959     \dim_gset:Nn \g_@@_blocks_ht_dim
7960     {

```

```

7961     \dim_max:nn
7962     \g_@@_blocks_ht_dim
7963     {
7964         \box_ht:c
7965         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7966     }
7967 }
7968 \dim_gset:Nn \g_@@_blocks_dp_dim
7969 {
7970     \dim_max:nn
7971     \g_@@_blocks_dp_dim
7972     {
7973         \box_dp:c
7974         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7975     }
7976 }
7977 }

7978 \cs_new:Npn \@@_adjust_hpos_rotate:
7979 {
7980     \bool_if:NT \g_@@_rotate_bool
7981     {
7982         \str_set:Nc \l_@@_hpos_block_str
7983         {
7984             \bool_if:NTF \g_@@_rotate_c_bool
7985             c
7986             {
7987                 \str_case:onF \l_@@_vpos_block_str
7988                 { b l B l t r T r }
7989                 {
7990                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7991                     r
7992                     l
7993                 }
7994             }
7995         }
7996     }
7997 }
7998 \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*.

```

7999 \cs_new_protected:Npn \@@_rotate_box_of_block:
8000 {
8001     \box_grotate:cn
8002     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8003     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8004     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8005     {
8006         \vbox_gset_top:cn
8007         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8008         {
8009             \skip_vertical:n { 0.8 ex }
8010             \box_use:c
8011             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8012         }
8013     }
8014     \bool_if:NT \g_@@_rotate_c_bool
8015     {
8016         \hbox_gset:cn
8017         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8018         {

```

```

8019 \IfFormatAtLeastTF { 2026-04-01}
8020 {
8021   \vbox_center:n
8022   {
8023     \box_use:c
8024     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8025   }
8026 }
8027 {
8028   $ % $
8029   \vcenter
8030   {
8031     \box_use:c
8032     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8033   }
8034   $ % $
8035 }
8036 }
8037 }
8038 }

```

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.

```

8039 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8040 {
8041   \seq_gput_right:Ne \g_@@_blocks_seq
8042   {
8043     \l_tmpa_tl
8044     { \exp_not:n { #3 } }
8045     {
8046       \bool_if:NTF \l_@@_tabular_bool
8047       {
8048

```

The following command will be no-op when respect-arraystretch is in force.

```

8049 \@@_reset_arraystretch:
8050 \exp_not:n
8051 {
8052   \dim_zero:N \extrarowheight
8053   #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).

```

8054 \tag_if_active:T { \tag_stop:n { table } }
8055 \use:e
8056 {
8057   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8058   { @ { } \l_@@_hpos_block_str @ { } }
8059 }
8060 #5
8061 \end { tabular }
8062 }
8063 }
8064 }

```


When we are *not* in an environment `{NiceTabular}` (or similar).

```

8065         {
8066         {
The following will be no-op when respect-arraystretch is in force.
8067             \@@_reset_arraystretch:
8068             \exp_not:n
8069             {
8070                 \dim_zero:N \extrarowheight
8071                 #4
8072                 $ % $
8073                 \use:e
8074                 {
8075                     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8076                     { @ { } \l_@@_hpos_block_str @ { } }
8077                 }
8078                 \@@_tuning_key_small: % 2026/05/04
8079                 #5
8080                 \end { array }
8081                 $ % $
8082             }
8083         }
8084     }
8085 }
8086 }
8087 }
8088 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8089 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8090 {
8091     \seq_gput_right:Ne \g_@@_blocks_seq
8092     {
8093         \l_tmpa_tl
8094         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8095         { { \exp_not:n { #4 #5 } } }
8096     }
8097 }
8098 \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`.

```

8099 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8100 {
8101     \seq_gput_right:Ne \g_@@_blocks_seq
8102     {
8103         \l_tmpa_tl
8104         { \exp_not:n { #3 } }
8105         { \exp_not:n { #4 #5 } }
8106     }
8107 }
8108 \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).

```

8109 \keys_define:nn { nicematrix / Block }
8110 {
8111     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8112     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8113   tikz .code:n =
8114       \IfPackageLoadedTF { tikz }
8115       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8116       { \@@_error:n { tikz~key~without~tikz } } ,
8117   tikz .value_required:n = true ,
8118   fill .code:n =
8119       \tl_set_rescan:Nnn
8120       \l_@@_fill_tl
8121       { \char_set_catcode_other:N ! }
8122       { #1 } ,
8123   fill .value_required:n = true ,

```

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

```

8124   opacity .tl_set:N = \l_@@_opacity_tl ,
8125   opacity .value_required:n = true ,
8126   draw .code:n =
8127       \tl_set_rescan:Nnn
8128       \l_@@_draw_tl
8129       { \char_set_catcode_other:N ! }
8130       { #1 } ,
8131   draw .default:n = default ,
8132   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8133   rounded-corners .default:n = 4 pt ,
8134   color .code:n =
8135       \@@_color:n { #1 }
8136       \tl_set_rescan:Nnn
8137       \l_@@_draw_tl
8138       { \char_set_catcode_other:N ! }
8139       { #1 } ,
8140   borders .clist_set:N = \l_@@_borders_clist ,
8141   borders .default:n = all , % added 2026/05/08
8142   hvlines .meta:n = { vlines , hlines } ,
8143   vlines .bool_set:N = \l_@@_vlines_block_bool ,
8144   hlines .bool_set:N = \l_@@_hlines_block_bool ,
8145   rules/width .dim_set:N = \arrayrulewidth ,
8146   rules/color .tl_set:N = \l_@@_rules_color_tl ,
8147   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8148   line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8149   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8150       \bool_set_true:N \l_@@_p_block_bool ,
8151   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8152   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8153   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8154   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8155       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8156   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8157       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8158   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8159       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8160   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8161   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8162   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8163   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8164   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8165   m .value_forbidden:n = true ,
8166   v-center .meta:n = m ,
8167   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8168   p .value_forbidden:n = true ,
8169   name .str_set:N = \l_@@_block_name_str ,
8170   name .value_required:n = true ,

```

```

8171   respect-arraystretch .code:n =
8172     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8173   respect-arraystretch .value_forbidden:n = true ,
8174   transparent .bool_set:N = \l_@@_transparent_bool ,
8175   unknown .code:n =
8176     \@@_unknown_key:nn
8177     { nicematrix / Block }
8178     { Unknown-key-for-Block }
8179 }

```

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.

```

8180 \cs_new_protected:Npn \@@_draw_blocks:
8181 {
8182   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8183   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8184 }
8185 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8186 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8187   \int_zero:N \l_@@_last_row_int
8188   \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”).

```

8189   \int_compare:nNnTF { #3 } > { 98 }
8190   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8191   { \int_set:Nn \l_@@_last_row_int { #3 } }
8192   \int_compare:nNnTF { #4 } > { 98 }
8193   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8194   { \int_set:Nn \l_@@_last_col_int { #4 } }
8195   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8196   {
8197     \bool_lazy_and:nnTF
8198     \l_@@_preamble_bool
8199     {
8200       \int_compare_p:n
8201       { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8202     }
8203     {
8204       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8205       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8206       \@@_msg_redirect_name:nn { columns-not-used } { none }
8207     }
8208     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8209   }
8210   {
8211     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8212     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8213     {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8214   \use:e
8215   {
8216     \@@_Block_v:nnnnnn

```

```

8217         { #1 }
8218         { #2 }
8219         { \int_use:N \l_@@_last_row_int }
8220         { \int_use:N \l_@@_last_col_int }
8221     }
8222     { #5 }
8223     { #6 }
8224 }
8225 }
8226 }

```

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.

```

8227 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8228 {

```

The group is for the keys.

```

8229     \group_begin:
8230     \int_compare:nNt { #1 } = { #3 }
8231     { \str_set:Nn \l_@@_vpos_block_str { t } }
8232     \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`.

```

8233     \bool_if:NT \l_@@_amp_in_blocks_bool
8234     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }
8235     \bool_lazy_and:nnT
8236     \l_@@_vlines_block_bool
8237     { ! \l_@@_ampersand_bool }
8238     {
8239         \tl_gput_right:Ne \g_@@_rules_tl
8240         {
8241             \@@_vlines_block:nnnnnn
8242             { \dim_use:N \arrayrulewidth }
8243             { \l_@@_rules_color_tl }
8244             { #1 } { #2 } { #3 } { #4 }
8245         }
8246     }
8247     \bool_if:NT \l_@@_hlines_block_bool
8248     {
8249         \tl_gput_right:Ne \g_@@_rules_tl
8250         {
8251             \@@_hlines_block:nnnnnn
8252             { \dim_use:N \arrayrulewidth }
8253             { \l_@@_rules_color_tl }
8254             { #1 } { #2 } { #3 } { #4 }
8255         }
8256     }

```

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).

```

8257     \bool_if:NF \l_@@_transparent_bool
8258     {
8259         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8260         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8261     }

8262     \tl_if_empty:NF \l_@@_draw_tl
8263     {
8264         \tl_gput_right:Nn \g_@@_rules_tl
8265         {
8266             \@@_stroke_block:nnnnn

```

#5 are the options

```

8267         { #5 } { #1 } { #2 } { #3 } { #4 }
8268     }
8269     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8270     { { #1 } { #2 } { #3 } { #4 } }
8271 }
8272 \clist_if_empty:NF \l_@@_borders_clist
8273 {
8274     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8275     {
8276         \@@_stroke_borders_block:nnnnn
8277         { #5 } { #1 } { #2 } { #3 } { #4 }
8278     }
8279 }
8280 \tl_if_empty:NF \l_@@_fill_tl
8281 {
8282     \@@_add_opacity_to_fill:
8283     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8284     {
8285         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8286         { #1 - #2 }
8287         { #3 - #4 }
8288         { \dim_use:N \l_@@_rounded_corners_dim }
8289     }
8290 }
8291 \seq_if_empty:NF \l_@@_tikz_seq
8292 {
8293     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8294     {
8295         \@@_block_tikz:nnnnn
8296         { \seq_use:Nn \l_@@_tikz_seq { , } }
8297         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8298     }
8299 }
8300 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8301 {
8302     \tl_gput_right:Ne \g_@@_rules_tl
8303     {
8304         \@@_draw_diagbox:nnnnnn
8305         { #1 } { #2 } { #3 } { #4 }
8306         { \exp_not:n { ##1 } }
8307         { \exp_not:n { ##2 } }
8308     }
8309 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & four & five & \\\
six & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

8310 \pgfpicture
8311 \pgfrememberpicturepositiononpagetrue
8312 \pgf@relevantforpicturesizefalse
8313 \@@_qpoint:n { row - #1 }
8314 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8315 \@@_qpoint:n { col - #2 }
8316 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8317 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8318 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8319 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8320 \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.

```

8321 \@@_pgf_rect_node:nnnnn
8322 { \@@_env: - #1 - #2 - block }
8323 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8324 \str_if_empty:NF \l_@@_block_name_str
8325 {
8326 \pgfnodealias
8327 { \@@_env: - \l_@@_block_name_str }
8328 { \@@_env: - #1 - #2 - block }
8329 \str_if_empty:NF \l_@@_name_str
8330 {
8331 \pgfnodealias
8332 { \l_@@_name_str - \l_@@_block_name_str }
8333 { \@@_env: - #1 - #2 - block }
8334 }
8335 }

```

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.

```

8336 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8337 {
8338 \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.

```

8339 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8340 {

```

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.

```

8341 \cs_if_exist:cT
8342 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8343 {
8344 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8345 {
8346 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8347 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8348 }
8349 }
8350 }

```

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.

```

8351 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8352 {
8353   \@@_qpoint:n { col - #2 }
8354   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8355 }
8356 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8357 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8358 {
8359   \cs_if_exist:cT
8360   { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8361   {
8362     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8363     {
8364       \pgfpointanchor
8365       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8366       { east }
8367       \dim_set:Nn \l_@@_tmpd_dim
8368       { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8369     }
8370   }
8371 }
8372 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8373 {
8374   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8375   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8376 }
8377 \@@_pgf_rect_node:nnnnn
8378 { \@@_env: - #1 - #2 - block - short }
8379 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8380 }

```

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.

```

8381 \bool_if:NT \l_@@_medium_nodes_bool
8382 {
8383   \@@_pgf_rect_node:nnn
8384   { \@@_env: - #1 - #2 - block - medium }
8385   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8386   {
8387     \pgfpointanchor
8388     { \@@_env:
8389       - \int_use:N \l_@@_last_row_int
8390       - \int_use:N \l_@@_last_col_int - medium
8391     }
8392     { south-east }
8393   }
8394 }
8395 \endpgfpicture
8396

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8397 \bool_if:NTF \l_@@_ampersand_bool
8398 {
8399   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8400   \int_zero_new:N \l_@@_split_int
8401   \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).

```

8402 \int_set:Nn \l_@@_first_row_int { #1 }

```

```

8403 \int_set:Nn \l_@@_first_col_int { #2 }
8404 \int_set:Nn \l_@@_last_row_int { #3 }
8405 \int_set:Nn \l_@@_last_col_int { #4 }

8406 \pgfpicture
8407 \pgfrememberpicturepositiononpagetrue
8408 \pgf@relevantforpicturesizefalse
8409 \@@_qpoint:n { row - #1 }
8410 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8411 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8412 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8413 \@@_qpoint:n { col - #2 }
8414 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8415 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8416 \dim_set:Nn \l_tmpb_dim
8417 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8418 \bool_lazy_or:nnT
8419 \l_@@_vlines_block_bool
8420 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8421 {
8422   \int_step_inline:nn { \l_@@_split_int - 1 }
8423   {
8424     \pgfpathmoveto
8425     {
8426       \pgfpoint
8427       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8428       \l_@@_tmpc_dim
8429     }
8430     \pgfpathlineto
8431     {
8432       \pgfpoint
8433       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8434       \l_@@_tmpd_dim
8435     }
8436     \CT@arc@
8437     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8438     \pgfsetrectcap
8439     \pgfusepathqstroke
8440   }
8441 }
8442 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8443 \int_zero_new:N \l_@@_split_i_int

8444 \str_if_eq:eeTF \l_@@_vpos_block_str T
8445 {
8446   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8447   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8448 }
8449 {
8450   \str_if_eq:eeTF \l_@@_vpos_block_str B
8451   {
8452     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8453     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8454   }
8455   {
8456     \bool_lazy_or:nnTF
8457     { \int_compare_p:nNn { #1 } = { #3 } }
8458     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8459     {
8460       \@@_qpoint:n { row - #1 - base }
8461       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8462     }
8463     {
8464       \str_if_eq:eeTF \l_@@_vpos_block_str b
8465       {

```



```

8466         \l_@@_qpoint:n { row - #3 - base }
8467         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8468     }
8469     {
8470         \l_@@_qpoint:n { #1 - #2 - block }
8471         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8472     }
8473 }
8474 }
8475 }
8476 \int_step_inline:nn \l_@@_split_int
8477 {
8478     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\l_@@_subcellcolor`.

```

8479     \int_set:Nn \l_@@_split_i_int { ##1 }
8480     \dim_set:Nn \col@sep
8481     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8482     \pgftransformshift
8483     {
8484         \pgfpoint
8485         {
8486             \l_tmpa_dim + ##1 \l_tmpb_dim -
8487             \str_case:on \l_@@_hpos_block_str
8488             {
8489                 l { \l_tmpb_dim + \col@sep }
8490                 c { 0.5 \l_tmpb_dim }
8491                 r { \col@sep }
8492             }
8493         }
8494         { \l_@@_tmpc_dim }
8495     }
8496     \pgfset { inner~sep = \c_zero_dim }
8497     \pgfnode
8498     { rectangle }
8499     {
8500         \str_if_eq:eeTF T \l_@@_vpos_block_str
8501         {
8502             \str_case:on \l_@@_hpos_block_str
8503             {
8504                 l { north-west }
8505                 c { north }
8506                 r { north-east }
8507             }
8508         }
8509         {
8510             \str_if_eq:eeTF B \l_@@_vpos_block_str
8511             {
8512                 \str_case:on \l_@@_hpos_block_str
8513                 {
8514                     l { south-west }
8515                     c { south }
8516                     r { south-east }
8517                 }
8518             }
8519         }
8520         \bool_lazy_any:nTF
8521         {
8522             { \int_compare_p:nNn { #1 } = { #3 } }
8523             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8524             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8525         }
8526         {
8527             \str_case:on \l_@@_hpos_block_str

```

```

8528             {
8529                 l { base~west }
8530                 c { base }
8531                 r { base~east }
8532             }
8533         }
8534     {
8535         \str_case:on \l_@@_hpos_block_str
8536         {
8537             l { west }
8538             c { center }
8539             r { east }
8540         }
8541     }
8542 }
8543 }
8544 }
8545 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8546 \group_end:
8547 }
8548 \endpgfpicture
8549 }

```

Now the case where there is no ampersand & in the content of the block.

```

8550 {
8551     \bool_if:NTF \l_@@_p_block_bool
8552     {

```

When the final user has used the key p, we have to compute the width.

```

8553     \pgfpicture
8554     \pgfrememberpicturepositiononpagetrue
8555     \pgf@relevantforpicturesizefalse
8556     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8557     {
8558         \@@_qpoint:n { col - #2 }
8559         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8560         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8561     }
8562     {
8563         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8564         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8565         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8566     }
8567     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8568 \endpgfpicture
8569 \hbox_set:Nn \l_@@_cell_box
8570 {
8571     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8572     { \g_tmpb_dim }
8573     \str_case:on \l_@@_hpos_block_str
8574     { c \centering r \raggedleft l \raggedright j { } }
8575     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8576     #6
8577     \end { minipage }
8578 }
8579 }
8580 {
8581     \hbox_set:Nn \l_@@_cell_box
8582     {
8583         \set@color
8584         \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8585         #6
8586     }
8587 }
8588 \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.

```

8589 \pgfpicture
8590 \pgfrememberpicturepositiononpagetrue
8591 \pgf@relevantforpicturesizefalse
8592 \bool_lazy_any:nTF
8593 {
8594   { \str_if_empty_p:N \l_@@_vpos_block_str }
8595   { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8596   { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8597   { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8598 }
8599 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8600 \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`.

```

8601 \bool_if:nT \g_@@_last_col_found_bool
8602 {
8603   \int_compare:nNnT { #2 } = \g_@@_col_total_int
8604   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8605 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8606 \tl_set:Ne \l_tmpa_tl
8607 {
8608   \str_case:on \l_@@_vpos_block_str
8609   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8610   { } {
8611     \str_case:on \l_@@_hpos_block_str
8612     {
8613       c { center }
8614       l { west }
8615       r { east }
8616       j { center }
8617     }
8618   }
8619   c {
8620     \str_case:on \l_@@_hpos_block_str
8621     {
8622       c { center }
8623       l { west }
8624       r { east }
8625       j { center }
8626     }
8627   }
8628   }
8629   T {
8630     \str_case:on \l_@@_hpos_block_str
8631     {
8632       c { north }
8633       l { north~west }
8634       r { north~east }
8635       j { north }
8636     }
8637   }
8638   }
8639   B {
8640     \str_case:on \l_@@_hpos_block_str

```

```

8641         {
8642             c { south }
8643             l { south~west }
8644             r { south~east }
8645             j { south }
8646         }
8647     }
8648 }
8649 }
8650 }
8651 \pgftransformshift
8652 {
8653     \pgfpointanchor
8654     {
8655         \@@_env: - #1 - #2 - block
8656         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8657     }
8658     \l_tmpa_tl
8659 }
8660 \pgfset { inner~sep = \c_zero_dim }
8661 \pgfnode
8662 { rectangle }
8663 \l_tmpa_tl
8664 { \box_use_drop:N \l_@@_cell_box } { } { } { }
8665 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8666 {
8667     \pgfextracty \l_tmpa_dim
8668     {
8669         \@@_qpoint:n
8670         {
8671             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8672             - base
8673         }
8674     }
8675     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8676 \pgfpointanchor
8677 {
8678     \@@_env: - #1 - #2 - block
8679     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8680 }
8681 {
8682     \str_case:on \l_@@_hpos_block_str
8683     {
8684         c { center }
8685         l { west }
8686         r { east }
8687         j { center }
8688     }
8689 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8690 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8691 \pgfset { inner~sep = \c_zero_dim }
8692 \pgfnode
8693 { rectangle }
8694 {
8695     \str_case:on \l_@@_hpos_block_str
8696     {
8697         c { base }
8698         l { base~west }

```

```

8699             r { base~east }
8700             j { base }
8701         }
8702     }
8703     { \box_use_drop:N \l_@@_cell_box } { } { }
8704 }
8705 \endpgfpicture
8706 }
8707 \group_end:
8708 }
8709 \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).

```

8710 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8711 {
8712     \tl_if_empty:NF \l_@@_opacity_tl
8713     {
8714         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8715             {
8716                 \tl_set:Nc \l_@@_fill_tl
8717                 {
8718                     [ opacity = \l_@@_opacity_tl ,
8719                     \tl_tail:o \l_@@_fill_tl
8720                 }
8721             }
8722             {
8723                 \tl_set:Nc \l_@@_fill_tl
8724                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8725             }
8726         }
8727     }

```

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*.

```

8728 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8729 {
8730     \group_begin:
8731     \tl_clear:N \l_@@_draw_tl
8732     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8733     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8734     \tl_if_empty:NF \l_@@_draw_tl
8735     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8736         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8737         \CT@arc@
8738         { \@@_color:o \l_@@_draw_tl }
8739     }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8740 \pgfsetcornersarced
8741 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8742 \int_compare:nNnF { #2 } > \c@iRow
8743 {
8744     \int_compare:nNnF { #3 } > \c@jCol
8745     {
8746         \@@_qpoint:n { row - #2 }
8747         \dim_set_eq:NN \l_tmpb_dim \pgf@y
8748         \@@_qpoint:n { col - #3 }

```

```

8749 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8750 \@@_qpoint:n
8751 { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8752 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8753 \@@_qpoint:n
8754 { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8755 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8756 \pgfpathrectanglecorners
8757 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8758 { \pgfpoint \pgf@x \l_tmpa_dim }
8759 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8760 \pgfusepathqstroke
8761 { \pgfusepath { stroke } }
8762 }
8763 }
8764 \group_end:
8765 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8766 \keys_define:nn { nicematrix / BlockStroke }
8767 {
8768   color .tl_set:N = \l_@@_draw_tl ,
8769   draw .code:n =
8770     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8771   draw .default:n = default ,
8772   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8773   rounded-corners .default:n = 4 pt ,
8774   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8775   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8776 }

```

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 *th* color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8777 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8778 {
8779   \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).

```

8780   \dim_set:Nn \arrayrulewidth { #1 }
8781   \pgfsetlinewidth \arrayrulewidth % added 2026-03-28
8782   \@@_set_CTarc:n { #2 }
8783   \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).

```

8784   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8785   {
8786     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8787     ||
8788     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8789     ||
8790     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8791     ||
8792     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8793   }

```

```

8794 \bool_if:NT \l_@@_fix_vertex_bool
8795 {
8796   \int_compare:nNtT { #4 } > 1
8797   {
8798     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8799     {
8800       { 1 }
8801       { 1 }
8802       { \int_use:N \c@iRow }
8803       { \int_eval:n { #4 -1 } }
8804       { }
8805     }
8806   }
8807   \int_compare:nNtT { #6 } < \c@jCol
8808   {
8809     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8810     {
8811       { 1 }
8812       { \int_eval:n { #6 + 1 } }
8813       { \int_use:N \c@iRow }
8814       { \int_use:N \c@jCol }
8815       { }
8816     }
8817   }
8818 }
8819 \int_set:Nn \l_@@_start_int { #3 }
8820 \int_set:Nn \l_@@_end_int { #5 }
8821 \int_step_inline:nnn { #4 } { #6 + 1 }
8822 {
8823   \int_set:Nn \l_@@_position_int { ##1 }
8824   \socket_use:n { nicematrix / draw-vrule }
8825 }
8826 \group_end:
8827 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8828 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8829 {
8830   \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).

```

8831   \dim_set:Nn \arrayrulewidth { #1 }
8832   \pgfsetlinewidth \arrayrulewidth % added 2026/03/28
8833   \@@_set_CTarc:n { #2 }
8834   \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).

```

8835   \seq_set_filter:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8836   {
8837     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8838     ||
8839     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8840     ||
8841     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8842     ||
8843     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8844   }

```

```

8845 \bool_if:NT \l_@@_fix_vertex_bool
8846 {
8847   \int_compare:nNnT { #3 } > 1
8848   {
8849     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8850     {
8851       { 1 }
8852       { 1 }
8853       { \int_eval:n { #3 - 1 } }
8854       { \int_use:N \c@jCol }
8855       { }
8856     }
8857   }
8858   \int_compare:nNnT { #5 } < \c@iRow
8859   {
8860     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8861     {
8862       { \int_eval:n { #5 + 1 } }
8863       { 1 }
8864       { \int_use:N \c@iRow }
8865       { \int_use:N \c@jCol }
8866       { }
8867     }
8868   }
8869 }
8870 \int_set:Nn \l_@@_start_int { #4 }
8871 \int_set:Nn \l_@@_end_int { #6 }
8872 \int_step_inline:nnn { #3 } { #5 + 1 }
8873 {
8874   \int_set:Nn \l_@@_position_int { ##1 }
8875   \socket_use:n { nicematrix / draw-hrule }
8876 }
8877 \group_end:
8878 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8879 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8880 {
8881   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8882   \keys_set:known:n { nicematrix / BlockBorders } { #1 }
8883
8884   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8885   { \@@_error:n { borders~forbidden } }
8886   {
8887     \tl_clear_new:N \l_@@_borders_tikz_tl
8888     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8889     \tl_set:Nn \l_@@_tmpc_tl { #2 }
8890     \tl_set:Nn \l_@@_tmpd_tl { #3 }
8891     \tl_set:Nn \l_tmpa_tl { \int_eval:n { #4 + 1 } }
8892     \tl_set:Nn \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8893     \@@_stroke_borders_block_i:
8894   }
8895 }
8896 \AtBeginDocument
8897 {
8898   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8899   {
8900     \c_@@_pgfortikzpicture_tl
8901     \@@_stroke_borders_block_ii:
8902     \c_@@_endpgfortikzpicture_tl
8903   }

```



```

8904 }
8905 \cs_new_protected:Npn \l_@@_stroke_borders_block_ii:
8906 {
8907   \pgfrememberpicturerepositiononpagetrue
8908   \pgf@relevantforpicturesizefalse
8909   \CT@arc@
8910   \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).

```

8911   \bool_set_false:N \l_tmpa_bool % added 2026/05/25
8912   \clist_if_in:NnT \l_@@_borders_clist { right }
8913   {
8914     \l_@@_stroke_vertical:n \l_tmpb_tl
8915     \bool_set_true:N \l_tmpa_bool
8916   }
8917   \clist_if_in:NnT \l_@@_borders_clist { left }
8918   {
8919     \l_@@_stroke_vertical:n \l_@@_tmpd_tl
8920     \bool_set_true:N \l_tmpa_bool
8921   }
8922   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8923   {
8924     \l_@@_stroke_horizontal:n \l_tmpa_tl
8925     \bool_set_true:N \l_tmpa_bool
8926   }
8927   \clist_if_in:NnT \l_@@_borders_clist { top }
8928   {
8929     \l_@@_stroke_horizontal:n \l_@@_tmpc_tl
8930     \bool_set_true:N \l_tmpa_bool
8931   }
8932   \bool_if:NF \l_tmpa_bool
8933   {
8934     \l_@@_stroke_vertical:n \l_tmpb_tl
8935     \l_@@_stroke_vertical:n \l_@@_tmpd_tl
8936     \l_@@_stroke_horizontal:n \l_tmpa_tl
8937     \l_@@_stroke_horizontal:n \l_@@_tmpc_tl
8938   }
8939 }
8940 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8941 {
8942   tikz .code:n =
8943     \cs_if_exist:NTF \tikzpicture
8944     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8945     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8946   tikz .value_required:n = true ,
8947   dashed .meta:n = { tikz = dashed } , % added 2026/05/25
8948   top .code:n = ,
8949   bottom .code:n = ,
8950   left .code:n = ,
8951   right .code:n = ,
8952   all .code:n = ,
8953   unknown .code:n = \@@_error:n { bad~border }
8954 }

```

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).

```

8955 \cs_new_protected:Npn \l_@@_stroke_vertical:n #1
8956 {
8957   \@@_qpoint:n \l_@@_tmpc_tl
8958   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8959   \@@_qpoint:n \l_tmpa_tl
8960   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }

```

```

8961 \@@_qpoint:n { #1 }
8962 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8963 {
8964   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8965   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8966   \pgfusepathqstroke
8967 }
8968 {
8969   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8970   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8971 }
8972 }

```

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).

```

8973 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8974 {
8975   \@@_qpoint:n \l_@@_tmpd_tl
8976   \clist_if_in:NnTF \l_@@_borders_clist { left }
8977   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8978   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8979   \@@_qpoint:n \l_tmpb_tl
8980   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8981   \@@_qpoint:n { #1 }
8982   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8983   {
8984     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8985     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8986     \pgfusepathqstroke
8987   }
8988   {
8989     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8990     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8991   }
8992 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8993 \keys_define:nn { nicematrix / BlockBorders }
8994 {
8995   borders .clist_set:N = \l_@@_borders_clist ,
8996   borders .default:n = all , % 2026/08/05
8997   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8998   rounded-corners .default:n = 4 pt ,
8999   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9000   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9001 }

```

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.

```

9002 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9003 {
9004   \begin { tikzpicture }
9005   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```
9006 \clist_map_inline:nn { #1 }
9007 {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
9008 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9009 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9010 (
9011 [
9012 xshift = \dim_use:N \l_@@_offset_dim ,
9013 yshift = - \dim_use:N \l_@@_offset_dim
9014 ]
9015 #2 -| #3
9016 )
9017 rectangle
9018 (
9019 [
9020 xshift = - \dim_use:N \l_@@_offset_dim ,
9021 yshift = \dim_use:N \l_@@_offset_dim
9022 ]
9023 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9024 ) ;
9025 }
9026 \end { tikzpicture }
9027 }
9028 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

9029 \keys_define:nn { nicematrix / SpecialOffset }
9030 {
9031 offset .dim_set:N = \l_@@_offset_dim ,
9032 }
```

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.

```
9033 \cs_new_protected:Npn \@@_NullBlock:
9034 { \@@_collect_options:n { \@@_NullBlock_i: } }
9035 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9036 { }
```

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.

```
9037 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9038 {
9039 \tl_gput_right:Ne \g_@@_pre_code_before_tl
9040 }
```

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`).

```
9041 \@@_subcellcolor:nnnnnnn
9042 {
9043 \tl_if_blank:nTF { #1 }
9044 { { \exp_not:n { #2 } } }
9045 { [ #1 ] { \exp_not:n { #2 } } }
9046 }
9047 { \int_use:N \l_@@_first_row_int } % first row of the block
9048 { \int_use:N \l_@@_first_col_int } % first column of the block
9049 { \int_use:N \l_@@_last_row_int } % last row of the block
9050 { \int_use:N \l_@@_last_col_int } % last column of the block
9051 { \int_use:N \l_@@_split_int }
9052 { \int_use:N \l_@@_split_i_int }
9053 }
```

```

9054 \ignorespaces
9055 }
9056 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9057 {
9058   \@@_color_opacity: #1
9059   \pgfpicture
9060   \pgf@relevantforpicturesizefalse
9061   \@@_qpoint:n { col - #3 }
9062   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9063   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9064   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9065   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9066   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9067   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9068   \@@_qpoint:n { row - #2 }
9069   \pgfpathrectanglecorners
9070     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9071     { \pgfpoint \l_tmpb_dim \pgf@y }
9072   \pgfusepathqfill
9073   \endpgfpicture
9074 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

9075 \keys_define:nn { nicematrix / Auto }
9076 {
9077   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9078   columns-type .value_required:n = true ,
9079   l .meta:n = { columns-type = l } ,
9080   r .meta:n = { columns-type = r } ,
9081   c .meta:n = { columns-type = c } ,
9082   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9083   delimiters / color .value_required:n = true ,
9084   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9085   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9086   delimiters .value_required:n = true ,
9087   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9088   rounded-corners .default:n = 4 pt
9089 }
9090 \NewDocumentCommand \AutoNiceMatrixWithDelims
9091 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9092 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
9093 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9094 {

```

The group is for the protection of the keys.

```

9095 \group_begin:
9096 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9097 \use:e
9098 {
9099   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9100     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9101     [ \exp_not:o \l_tmpa_tl ]
9102   }
9103   \int_if_zero:nT \l_@@_first_row_int
9104   {
9105     \int_if_zero:nT \l_@@_first_col_int { & }

```

```

9106     \prg_replicate:nn { #4 - 1 } { & }
9107     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9108   }
9109   \prg_replicate:nn { #3 }
9110   {
9111     \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).

```

9112     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9113     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9114   }
9115   \int_compare:nNnT \l_@@_last_row_int > { -2 }
9116   {
9117     \int_if_zero:nT \l_@@_first_col_int { & }
9118     \prg_replicate:nn { #4 - 1 } { & }
9119     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9120   }
9121   \end { NiceArrayWithDelims }
9122   \group_end:
9123 }

9124 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9125 {
9126   \cs_set_protected:cpn { #1 AutoNiceMatrix }
9127   {
9128     \bool_gset_true:N \g_@@_delims_bool
9129     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
9130     \AutoNiceMatrixWithDelims { #2 } { #3 }
9131   }
9132 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

9133 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
9134 {
9135   \group_begin:
9136   \bool_gset_false:N \g_@@_delims_bool
9137   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9138   \group_end:
9139 }

```

28 The redefinition of the command \dotfill

```

9140 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9141 \cs_new_protected:Npn \@@_dotfill:
9142 {

```

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}).

```

9143   \@@_old_dotfill:
9144   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9145 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert \@@_dotfill (which is the saved version of \dotfill) in the cell of the array, and it will extend, since it is no longer in \l_@@_cell_box.

```

9146 \cs_new_protected:Npn \@@_dotfill_i:
9147 {
9148   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim

```

```

9149     { \@@_old_dotfill: }
9150 }

```

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.

```

9151 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9152 {
9153   \tl_gput_right:Ne \g_@@_rules_tl
9154   {
9155     \@@_draw_diagbox:nnnnnn
9156     { \int_use:N \c@iRow }
9157     { \int_use:N \c@jCol }
9158     { \int_use:N \c@iRow }
9159     { \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.

```

9160     { \g_@@_row_style_tl \exp_not:n { #1 } }
9161     { \g_@@_row_style_tl \exp_not:n { #2 } }
9162   }

```

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`.

```

9163   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9164   {
9165     { \int_use:N \c@iRow }
9166     { \int_use:N \c@jCol }
9167     { \int_use:N \c@iRow }
9168     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9169     { }
9170   }
9171 }

```

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.

```

9172 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9173 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9174   \begin { pgfscope }
9175   \@@_qpoint:n { row - #1 }
9176   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9177   \@@_qpoint:n { col - #2 }
9178   \dim_set_eq:NN \l_tmpb_dim \pgf@x
9179   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9180   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9181   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9182   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9183   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9184   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9185   {

```

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.

```

9186     \CT@arc@
9187     \pgfsetroundcap
9188     \pgfusepathqstroke
9189   }
9190   \pgfset { inner~sep = 1 pt }
9191   \pgfscope
9192   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9193   \pgfnode { rectangle } { south~west }
9194   {
9195     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9196     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9197     \end { minipage }
9198   }
9199   { }
9200   { }
9201 \endpgfscope
9202 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9203 \pgfnode { rectangle } { north~east }
9204 {
9205   \begin { minipage } { 20 cm }
9206   \raggedleft
9207   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9208   \end { minipage }
9209 }
9210 { }
9211 { }
9212 \end { pgfscope }
9213 }

```

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. 89.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9214 \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 `\`.

```

9215 \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`.

```

9216 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9217 {
9218   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9219   \@@_CodeAfter_iv:n
9220 }

```

We catch the argument of the command `\end` (in #1).

```

9221 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9222 {

```

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.

```
9223 \str_if_eq:eeTF \@currenvir { #1 }
9224 { \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`.

```
9225 {
9226 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9227 \@@_CodeAfter_ii:n
9228 }
9229 }
```

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.

```
9230 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9231 {
9232 \pgfpicture
9233 \pgfrememberpicturepositiononpagetrue
9234 \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.

```
9235 \@@_qpoint:n { row - 1 }
9236 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9237 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9238 \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).

```
9239 \bool_if:nTF { #3 }
9240 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9241 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9242 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9243 {
9244 \cs_if_exist:cT
9245 { pgf @ sh @ ns @ \@_env: - ##1 - #2 }
9246 {
9247 \pgfpointanchor
9248 { \@_env: - ##1 - #2 }
9249 { \bool_if:nTF { #3 } { west } { east } }
9250 \dim_set:Nn \l_tmpa_dim
9251 {
9252 \bool_if:nTF { #3 }
9253 \dim_min:nn
9254 \dim_max:nn
9255 \l_tmpa_dim
9256 \pgf@x
```



```

9257     }
9258   }
9259 }

```

Now we can put the delimiter with a node of PGF.

```

9260 \pgfset { inner~sep = \c_zero_dim }
9261 \dim_zero:N \nulldelimiterspace
9262 \pgftransformshift
9263 {
9264   \pgfpoint
9265     \l_tmpa_dim
9266     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9267 }
9268 \pgfnode
9269 { rectangle }
9270 { \bool_if:nTF { #3 } { east } { west } }
9271 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9272   \nullfont
9273   $ % $
9274   \@@_color:o \l_@@_delimiters_color_tl
9275   \bool_if:nTF { #3 } { \left #1 } { \left . }
9276   \vcenter
9277   {
9278     \nullfont
9279     \hrule \@height
9280       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9281       \@depth \c_zero_dim
9282       \@width \c_zero_dim
9283   }
9284   \bool_if:nTF { #3 } { \right . } { \right #1 }
9285   $ % $
9286 }
9287 { }
9288 { }
9289 \endpgfpicture
9290 }

```

32 The command \SubMatrix

```

9291 \keys_define:nn { nicematrix / sub-matrix }
9292 {
9293   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9294   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9295   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9296   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9297   xshift .value_required:n = true ,
9298   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9299   delimiters / color .value_required:n = true ,
9300   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9301   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9302   hlines .default:n = all ,
9303   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9304   vlines .default:n = all ,
9305   hvlines .meta:n = { hlines, vlines } ,
9306   hvlines .value_forbidden:n = true
9307 }
9308 \keys_define:nn { nicematrix }
9309 {
9310   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

9311 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9312 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9313 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9314 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9315 \keys_define:nn { nicematrix / SubMatrix }
9316 {
9317   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9318   delimiters / color .value_required:n = true ,
9319   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9320   hlines .default:n = all ,
9321   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9322   vlines .default:n = all ,
9323   hvlines .meta:n = { hlines, vlines } ,
9324   hvlines .value_forbidden:n = true ,
9325   name .code:n =
9326     \tl_if_empty:nTF { #1 }
9327     { \@@_error:n { Invalid-name } }
9328     {
9329       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9330       {
9331         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9332         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
9333         {
9334           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9335           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9336         }
9337       }
9338       { \@@_error:n { Invalid-name } }
9339     } ,
9340   name .value_required:n = true ,
9341   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9342   rules .value_required:n = true ,
9343   code .tl_set:N = \l_@@_code_tl ,
9344   code .value_required:n = true ,
9345   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9346 }

9347 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9348 {
9349   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9350   {
9351     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9352     [
9353       delimiters / color = \l_@@_delimiters_color_tl ,
9354       hlines = \l_@@_submatrix_hlines_clist ,
9355       vlines = \l_@@_submatrix_vlines_clist ,
9356       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9357       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9358       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9359       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9360       #5
9361     ]
9362   }
9363   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9364   \ignorespaces
9365 }

9366 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9367 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9368 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

```

```

9369 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9370 {
9371   \seq_gput_right:Ne \g_@@_submatrix_seq
9372   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9373     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9374     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9375     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9376     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9377   }
9378 }

```

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).

```

9379 \NewDocumentCommand \@@_compute_i_j:nn
9380 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9381 { \@@_compute_i_j:nnnn #1 #2 }

```

```

9382 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9383 {
9384   \def \l_@@_first_i_tl { #1 }
9385   \def \l_@@_first_j_tl { #2 }
9386   \def \l_@@_last_i_tl { #3 }
9387   \def \l_@@_last_j_tl { #4 }
9388   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9389     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9390   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9391     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9392   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9393     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9394   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9395     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9396 }

```

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`.

```

9397 \AtBeginDocument
9398 {
9399   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9400   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9401     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9402 }
9403 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9404 {
9405   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9406 \@@_compute_i_j:nn { #2 } { #3 }
9407 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
9408 { \def \arraystretch { 1 } }
9409 \bool_lazy_or:nnTF
9410 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9411 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9412 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
9413 {
9414 \str_clear_new:N \l_@@_submatrix_name_str
9415 \keys_set:nn { nicematrix / SubMatrix } { #5 }
9416 \pgfpicture
9417 \pgfrememberpicturepositiononpagetrue
9418 \pgf@relevantforpicturesizefalse
9419 \pgfset { inner~sep = \c_zero_dim }
9420 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9421 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9422 \bool_if:NTF \l_@@_submatrix_slim_bool
9423 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9424 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9425 {
9426 \cs_if_exist:cT
9427 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9428 {
9429 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9430 \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9431 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9432 }
9433 \cs_if_exist:cT
9434 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9435 {
9436 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9437 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9438 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9439 }
9440 }
9441 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9442 { \@@_impossible_delimiter:n { left } }
9443 {
9444 \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9445 { \@@_impossible_delimiter:n { right } }
9446 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9447 }
9448 \endpgfpicture
9449 }
9450 \group_end:
9451 \ignorespaces
9452 }

```

The argument of the following command will be provided by curryfication.

```

9453 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9454 {
9455 \bool_if:NTF \l_@@_no_cell_nodes_bool
9456 { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9457 { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9458 }

```

`#1` is the left delimiter, `#2` is the right one, `#3` is the subscript and `#4` is the superscript.

```

9459 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9460 {

```

```

9461 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9462 \dim_set:Nn \l_@@_y_initial_dim
9463 {
9464   \fp_to_dim:n
9465   {
9466     \pgf@y
9467     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9468   }
9469 }
9470 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9471 \dim_set:Nn \l_@@_y_final_dim
9472 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9473 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9474 {
9475   \cs_if_exist:cT
9476   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9477   {
9478     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9479     \dim_set:Nn \l_@@_y_initial_dim
9480     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9481   }
9482   \cs_if_exist:cT
9483   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9484   {
9485     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9486     \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9487     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9488   }
9489 }
9490 \dim_set:Nn \l_tmpa_dim
9491 {
9492   \l_@@_y_initial_dim - \l_@@_y_final_dim +
9493   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9494 }
9495 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9496 \group_begin:
9497 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9498 \@@_set_CTarc:o \l_@@_rules_color_tl
9499 \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`.

```

9500 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9501 {
9502   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9503   {
9504     \int_compare:nNnT
9505     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9506     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9507 \@@_qpoint:n { col - ##1 }
9508 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9509 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9510 \pgfusepathqstroke
9511 }
9512 }
9513 }

```

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.

```

9514 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9515 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9516 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9517 {
9518   \bool_lazy_and:nnTF
9519   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9520   {
9521     \int_compare_p:nNn
9522     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9523   {
9524     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9525     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9526     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9527     \pgfusepathqstroke
9528   }
9529   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9530 }

```

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.

```

9531 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9532 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9533 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9534 {
9535   \bool_lazy_and:nnTF
9536   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9537   {
9538     \int_compare_p:nNn
9539     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9540   {
9541     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9542   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
9543   \dim_set:Nn \l_tmpa_dim
9544   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9545   \str_case:nn { #1 }
9546   {
9547     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9548     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9549     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9550   }
9551   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

9552   \dim_set:Nn \l_tmpb_dim
9553   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9554   \str_case:nn { #2 }
9555   {
9556     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9557     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9558     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9559   }
9560   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9561   \pgfusepathqstroke
9562   \group_end:
9563 }
9564 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9565 }

```

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).

```

9566 \str_if_empty:NF \l_@@_submatrix_name_str
9567 {
9568   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9569   \l_@@_x_initial_dim \l_@@_y_initial_dim
9570   \l_@@_x_final_dim \l_@@_y_final_dim
9571 }
9572 \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`.

```

9573 \begin { pgfscope }
9574 \pgftransformshift
9575 {
9576   \pgfpoint
9577   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9578   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9579 }
9580 \str_if_empty:NTF \l_@@_submatrix_name_str
9581 { \@@_node_left:nn #1 { } }
9582 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9583 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9584 \pgftransformshift
9585 {
9586   \pgfpoint
9587   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9588   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9589 }
9590 \str_if_empty:NTF \l_@@_submatrix_name_str
9591 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9592 {
9593   \@@_node_right:nnnn #2
9594   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9595 }

```

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`.

```

9596 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9597 \flag_clear_new:N \l_@@_code_flag
9598 \l_@@_code_tl
9599 }

```

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`.

```

9600 \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.

```

9601 \cs_new:Npn \@@_pgfpointanchor:n #1
9602 { \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`).

```

9603 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9604 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9605 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9606 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9607 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9608 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9609 { \@@_pgfpointanchor_ii:n { #1 } }
9610 }

```

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`).

```

9611 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9612 { \@@_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`.

```

9613 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9614 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9615 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9616 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9617 { \@@_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`).

```

9618 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9619 }

```

The following function is for the case when the name contains an hyphen.

```

9620 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9621 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9622 \@@_env:
9623 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9624 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9625 }

```

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.

```

9626 \tl_const:Nn \c_@@_integers_alist_tl
9627 {
9628 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9629 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9630 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9631 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9632 }

```



```

9633 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9634 {

```

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`.

```

9635 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9636 {
9637 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9638 \@@_env: -
9639 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9640 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9641 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9642 }
9643 {
9644 \str_if_eq:eeTF { #1 } { last }
9645 {
9646 \flag_raise:N \l_@@_code_flag
9647 \@@_env: -
9648 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9649 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9650 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9651 }
9652 { #1 }
9653 }
9654 }

```

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`).

```

9655 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9656 {
9657 \pgfnode
9658 { rectangle }
9659 { east }
9660 {
9661 \nullfont
9662 $ % $
9663 \@@_color:o \l_@@_delimiters_color_tl
9664 \left #1
9665 \vcenter
9666 {
9667 \nullfont
9668 \hrule \@height \l_tmpa_dim
9669 \@depth \c_zero_dim
9670 \@width \c_zero_dim
9671 }
9672 \right .
9673 $ % $
9674 }
9675 { #2 }
9676 { }
9677 }

```

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.

```

9678 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9679 {
9680   \pgfnode
9681     { rectangle }
9682     { west }
9683     {
9684       \nullfont
9685       $ % $
9686       \colorlet { current-color } { . }
9687       \@@_color:o \l_@@_delimiters_color_tl
9688       \left .
9689       \vcenter
9690         {
9691           \nullfont
9692           \hrule \@height \l_tmpa_dim
9693             \@depth \c_zero_dim
9694             \@width \c_zero_dim
9695         }
9696       \right #1
9697       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9698       ~ { \color { current-color } \smash { #4 } }
9699       $ % $
9700     }
9701     { #2 }
9702     { }
9703 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9704 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9705 {
9706   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9707   \ignorespaces
9708 }
9709 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9710 {
9711   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9712   \ignorespaces
9713 }
9714 \keys_define:nn { nicematrix / Brace }
9715 {
9716   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9717   left-shorten .value_forbidden:n = true ,
9718   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9719   right-shorten .value_forbidden:n = true ,
9720   shorten .meta:n = { left-shorten , right-shorten } ,
9721   shorten .value_forbidden:n = true ,
9722   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9723   color .tl_set:N = \l_tmpa_tl ,
9724   color .value_required:n = true ,
9725   unknown .code:n =
9726     \@@_unknown_key:nn
9727     { nicematrix / Brace }

```

```

9728     { Unknown-key-for-Brace }
9729 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to `under` or `over`.

```

9730 \cs_new_protected:Npn \l_@@_brace:nnnnn #1 #2 #3 #4 #5
9731 {
9732   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9733   \l_@@_compute_i_j:nn { #1 } { #2 }
9734   \bool_lazy_or:nnTF
9735     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9736     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9737   {
9738     \str_if_eq:eeTF { #5 } { under }
9739     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9740     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9741   }
9742   {
9743     \tl_clear:N \l_tmpa_tl
9744     \keys_set:nn { nicematrix / Brace } { #4 }
9745     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9746     \pgfpicture
9747     \pgfrememberpicturepositiononpagetrue
9748     \pgf@relevantforpicturesizefalse
9749     \bool_if:NT \l_@@_brace_left_shorten_bool
9750     {
9751       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9752       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9753       {
9754         \cs_if_exist:cT
9755           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9756         {
9757           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9758
9759           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9760             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9761         }
9762       }
9763     }
9764     \bool_lazy_or:nnT
9765       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9766       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9767     {
9768       \@@_qpoint:n { col - \l_@@_first_j_tl }
9769       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9770     }
9771     \bool_if:NT \l_@@_brace_right_shorten_bool
9772     {
9773       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9774       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9775       {
9776         \cs_if_exist:cT
9777           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9778         {
9779           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9780           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9781             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9782         }
9783       }
9784     }
9785     \bool_lazy_or:nnT

```

```

9786 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9787 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9788 {
9789   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9790   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9791 }
9792 \pgfset { inner~sep = \c_zero_dim }
9793 \str_if_eq:eeTF { #5 } { under }
9794 { \@@_underbrace_i:n { #3 } }
9795 { \@@_overbrace_i:n { #3 } }
9796 \endpgfpicture
9797 }
9798 \group_end:
9799 }

```

The argument is the text to put above the brace.

```

9800 \cs_new_protected:Npn \@@_overbrace_i:n #1
9801 {
9802   \@@_qpoint:n { row - \l_@@_first_i_tl }
9803   \pgftransformshift
9804   {
9805     \pgfpoint
9806     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9807     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9808   }
9809   \pgfnode
9810   { rectangle }
9811   { south }
9812   {
9813     \vtop
9814     {
9815       \group_begin:
9816       \everycr { }
9817       \halign
9818       {
9819         \hfil ## \hfil \crcr
9820         \bool_if:NTF \l_@@_tabular_bool
9821         { \begin { tabular } { c } #1 \end { tabular } }
9822         { $ \begin { array } { c } #1 \end { array } $ }
9823         \cr
9824         $ % $
9825         \overbrace
9826         {
9827           \hbox_to_wd:nn
9828           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9829           { }
9830         }
9831         $ % $
9832         \cr
9833       }
9834       \group_end:
9835     }
9836   }
9837   { }
9838   { }
9839 }

```

The argument is the text to put under the brace.

```

9840 \cs_new_protected:Npn \@@_underbrace_i:n #1
9841 {
9842   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9843   \pgftransformshift
9844   {

```

```

9845     \pgfpoint
9846     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9847     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9848   }
9849   \pgfnode
9850   { rectangle }
9851   { north }
9852   {
9853     \group_begin:
9854     \everycr { }
9855     \vbox
9856     {
9857       \halign
9858       {
9859         \hfil ## \hfil \crcr
9860         $ % $
9861         \underbrace
9862         {
9863           \hbox_to_wd:nn
9864           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9865           { }
9866         }
9867         $ % $
9868         \cr
9869         \bool_if:NTF \l_@@_tabular_bool
9870         { \begin { tabular } { c } #1 \end { tabular } }
9871         { $ \begin { array } { c } #1 \end { array } $ }
9872         \cr
9873       }
9874     }
9875     \group_end:
9876   }
9877   { }
9878   { }
9879 }

```

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`).

```

9880 \AddToHook { package / tikz / after }
9881 {
9882   \tikzset
9883   {
9884     nicematrix / brace / .style =
9885     {
9886       decoration = { brace , raise = -0.15 em } ,
9887       decorate ,
9888     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9889     nicematrix / mirrored-brace / .style =
9890     {
9891       nicematrix / brace ,
9892       decoration = mirror ,

```

```

9893     }
9894   }
9895 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9896 \keys_define:nn { nicematrix / Hbrace }
9897 {
9898   color .code:n = ,
9899   horizontal-label .code:n = ,
9900   horizontal-labels .code:n = ,
9901   shorten .code:n = ,
9902   shorten-start .code:n = ,
9903   shorten-end .code:n = ,
9904   shorten+ .code:n = ,
9905   shorten-start+ .code:n = ,
9906   shorten-end+ .code:n = ,
9907   shorten~+ .code:n = ,
9908   shorten-start~+ .code:n = ,
9909   shorten-end~+ .code:n = ,
9910   brace-shift .code:n = ,
9911   brace-shift+ .code:n = ,
9912   brace-shift~+ .code:n = ,
9913   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9914 }

```

Here we need an “fully expandable” command.

```

9915 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9916 {
9917   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9918     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9919     { \@@_error:nn { Hbrace-not~allowed } { \Hbrace } }
9920 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9921 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9922 {
9923   \int_compare:nNnTF \c@iRow < { 2 }
9924   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9925     \str_if_eq:nnTF { #2 } { * }
9926     {
9927       \bool_set_true:N \l_@@_nullify_dots_bool
9928       \Ldots
9929       [
9930         line-style = nicematrix / brace ,
9931         #1 ,
9932         up =
9933         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9934       ]
9935     }
9936     {
9937       \Hdotsfor
9938       [
9939         line-style = nicematrix / brace ,
9940         #1 ,
9941         up =
9942         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9943       ]
9944       { #2 }

```

```

9945     }
9946   }
9947   {
9948     \str_if_eq:nnTF { #2 } { * }
9949     {
9950       \bool_set_true:N \l_@@_nullify_dots_bool
9951       \Ldots
9952       [
9953         line-style = nicematrix / mirrored-brace ,
9954         #1 ,
9955         down =
9956           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9957       ]
9958     }
9959     {
9960       \Hdotsfor
9961       [
9962         line-style = nicematrix / mirrored-brace ,
9963         #1 ,
9964         down =
9965           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9966       ]
9967       { #2 }
9968     }
9969   }
9970   \keys_set:nn { nicematrix / Hbrace } { #1 }
9971 }

```

```

9972 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9973 {
9974   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9975   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9976   { \@@_error:nn { Hbrace-not-allowed } { \Vbrace } }
9977 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9978 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9979 {
9980   \int_compare:nNnTF \c@jCol < { 2 }
9981   {
9982     \str_if_eq:nnTF { #2 } { * }
9983     {
9984       \bool_set_true:N \l_@@_nullify_dots_bool
9985       \Vdots
9986       [
9987         Vbrace ,
9988         line-style = nicematrix / mirrored-brace ,
9989         #1 ,
9990         down =
9991           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9992       ]
9993     }
9994     {
9995       \Vdotsfor
9996       [
9997         Vbrace ,
9998         line-style = nicematrix / mirrored-brace ,
9999         #1 ,
10000         down =
10001           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10002       ]
10003       { #2 }

```

```

10004     }
10005   }
10006   {
10007     \str_if_eq:nnTF { #2 } { * }
10008     {
10009       \bool_set_true:N \l_@@_nullify_dots_bool
10010       \Vdots
10011       [
10012         Vbrace ,
10013         line-style = nicematrix / brace ,
10014         #1 ,
10015         up =
10016         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10017       ]
10018     }
10019     {
10020       \Vdotsfor
10021       [
10022         Vbrace ,
10023         line-style = nicematrix / brace ,
10024         #1 ,
10025         up =
10026         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10027       ]
10028       { #2 }
10029     }
10030   }
10031   \keys_set:nn { nicematrix / Hbrace } { #1 }
10032 }

```

35 The command TikzEveryCell

```

10033 \bool_new:N \l_@@_not_empty_bool
10034 \bool_new:N \l_@@_empty_bool
10035
10036 \keys_define:nn { nicematrix / TikzEveryCell }
10037 {
10038   not-empty .code:n =
10039     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10040     { \bool_set_true:N \l_@@_not_empty_bool }
10041     { \@@_error:n { detection-of-empty-cells } } ,
10042   not-empty .value_forbidden:n = true ,
10043   empty .code:n =
10044     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10045     { \bool_set_true:N \l_@@_empty_bool }
10046     { \@@_error:n { detection-of-empty-cells } } ,
10047   empty .value_forbidden:n = true ,
10048   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10049 }
10050
10051
10052 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10053 {
10054   \IfPackageLoadedTF { tikz }
10055   {
10056     \group_begin:
10057     \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.


```

10058     \tl_set:Nn \l_tmpa_tl { { #2 } }
10059     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10060       { \@@_for_a_block:nnnnn #1 }
10061     \@@_all_the_cells:
10062     \group_end:
10063   }
10064   { \@@_error:n { TikzEveryCell~without~tikz } }
10065 }
10066
10067
10068 \cs_new_protected:Nn \@@_all_the_cells:
10069 {
10070   \int_step_inline:nn \c@iRow
10071   {
10072     \int_step_inline:nn \c@jCol
10073     {
10074       \cs_if_exist:cF { cell - ##1 - #####1 }
10075       {
10076         \clist_if_in:NcF \l_@@_corners_cells_clist
10077           { ##1 - #####1 }
10078         {
10079           \bool_set_false:N \l_tmpa_bool
10080           \cs_if_exist:cTF
10081             { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10082             {
10083               \bool_if:NF \l_@@_empty_bool
10084               { \bool_set_true:N \l_tmpa_bool }
10085             }
10086             {
10087               \bool_if:NF \l_@@_not_empty_bool
10088               { \bool_set_true:N \l_tmpa_bool }
10089             }
10090           \bool_if:NT \l_tmpa_bool
10091           {
10092             \@@_block_tikz:nnnnn
10093             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10094           }
10095         }
10096       }
10097     }
10098   }
10099 }
10100
10101 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10102 {
10103   \bool_if:NF \l_@@_empty_bool
10104   {
10105     \@@_block_tikz:nnnnn
10106     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10107   }
10108   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10109 }
10110
10111 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10112 {
10113   \int_step_inline:nnn { #1 } { #3 }
10114   {
10115     \int_step_inline:nnn { #2 } { #4 }
10116     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10117   }
10118 }

```

36 The key draw-trees-in-col

```

10119 \cs_new_protected:Npn \@@_draw_trees:
10120 {
10121   \bool_if:NTF \l_@@_no_cell_nodes_bool
10122     { \@@_error:n { draw-trees~with~no~cell~nodes } }
10123   \@@_draw_trees_i:
10124 }
10125 \cs_new_protected:Npn \@@_draw_trees_i:
10126 {
10127   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10128   \dim_zero_new:N \l_@@_em_dim
10129   \dim_set:Nn \l_@@_em_dim { 1 em }
10130   \dim_zero_new:N \l_@@_ex_dim
10131   \dim_set:Nn \l_@@_ex_dim { 1 ex }
10132   \pgfpicture
10133   \pgfrememberpicturepositiononpagetrue
10134   \pgf@relevantforpicturesizefalse
10135   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10136     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10137   \@@_color:o \l_@@_trees_color_tl
10138   \pgfsetcornersarced
10139   { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10140   \clist_map_function:NN \g_@@_col_with_trees_clist
10141     \@@_draw_trees_in_col:n
10142   \clist_gclear:N \g_@@_col_with_trees_clist
10143   \endpgfpicture
10144 }
10145 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10146 {

```

The argument is provided by curryfication.

```

10147   \int_compare:nNnTF { #1 } > \c@jCol
10148     { \@@_error:nn { Col~outside~tabular~in~trees } }
10149     {
10150       \int_compare:nNnTF { #1 } = \c@jCol
10151         { \@@_error:nn { Last~col~in~trees } }
10152         \@@_draw_trees_in_col_i:n
10153     }
10154   { #1 }
10155 }
10156 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10157 {
10158   \int_set:Nn \l_tmpa_int { 1 }
10159   \int_step_inline:nn { \c@iRow + 1 }
10160   {
10161     \cs_if_exist:cT
10162       { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10163       {
10164         \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10165         {

```

Now, you will, potentially, draw a tree.

```

10166       \@@_draw_tree:nee
10167       { #1 }
10168       { \int_use:N \l_tmpa_int }
10169       { \int_eval:n { ##1 - 1 } }
10170     }
10171   \int_set:Nn \l_tmpa_int { ##1 }
10172 }

```

```

10173     }
10174     \int_compare:nNnT \c@iRow > \l_tmpa_int
10175     {
10176         \@@_draw_tree:nee
10177         { #1 }
10178         { \int_use:N \l_tmpa_int }
10179         { \int_eval:n { \c@iRow } }
10180     }
10181 }

```

#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.

```

10182 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10183 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10184     \pgfpointanchor { \@@_env: - #1 } { 5 }
10185     \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.

```

10186     \bool_set_false:N \l_tmpa_bool
10187     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10188     {
10189         \cs_if_exist:cT
10190         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10191     {
10192         \pgfpointanchor
10193         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10194         { base~west }
10195         \pgfpathmoveto
10196         {
10197             \pgfpoint
10198             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10199             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10200         }
10201         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10202         \bool_if:NF \l_tmpa_bool
10203         {
10204             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10205             \pgfpathlineto
10206             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10207             \bool_set_true:N \l_tmpa_bool
10208         }
10209         \pgfusepath { stroke }
10210     }
10211 }
10212 }
10213 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10214 \cs_new_protected:Npn \@@_create_blocks_in_col:
10215 {
10216     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10217     \clist_map_inline:Nn \g_@@_cbic_clist
10218     {
10219         \cs_set:cpn
10220         {
10221             pgf @ sh @ ns @ \@@_env:
10222             - \int_eval:n { \c@iRow + 1 } - ##1 }

```

```

10223         { rien }

\l_tmpa_int will be the first row of the block being created.
10224     \int_set:Nn \l_tmpa_int { 1 }
10225     \int_step_inline:nn { \c@iRow + 1 }
10226     {
10227         \cs_if_exist:cT
10228         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10229         {
10230             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10231             {
10232                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10233                 {
10234                     { \int_use:N \l_tmpa_int }
10235                     { ##1 }
10236                     { \int_eval:n { #####1 - 1 } }
10237                     { ##1 }
10238                     { }
10239                 }
10240             }
10241             \int_set:Nn \l_tmpa_int { #####1 }
10242         }
10243     }
10244 }
10245 \clist_gclear:N \g_@@_cbic_clist
10246 }

```

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.

```

10247 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10248 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10249 \NewDocumentCommand \@@_ShowCellNames { }
10250 {
10251     \bool_if:NT \l_@@_in_code_after_bool
10252     {
10253         \pgfpicture
10254         \pgfrememberpicturepositiononpagetrue
10255         \pgf@relevantforpicturesizefalse
10256         \pgfpathrectanglecorners
10257         { \@@_qpoint:n { 1 } }
10258         { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10259         \pgfsetfillopacity { 0.75 }
10260         \pgfsetfillcolor { white }
10261         \pgfusepathqfill
10262         \endpgfpicture
10263     }
10264     \dim_gzero_new:N \g_@@_tmpc_dim
10265     \dim_gzero_new:N \g_@@_tmpd_dim
10266     \dim_gzero_new:N \g_@@_tmpe_dim
10267     \int_step_inline:nn \c@iRow
10268     {
10269         \bool_if:NTF \l_@@_in_code_after_bool
10270         {
10271             \pgfpicture
10272             \pgfrememberpicturepositiononpagetrue

```

```

10273     \pgf@relevantforpicturesizefalse
10274   }
10275   { \begin { pgfpicture } }
10276   \@@_qpoint:n { row - ##1 }
10277   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10278   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10279   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10280   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10281   \bool_if:NTF \l_@@_in_code_after_bool
10282     { \endpgfpicture }
10283     { \end { pgfpicture } }
10284   \int_step_inline:nn \c@jCol
10285   {
10286     \hbox_set:Nn \l_tmpa_box
10287     {
10288       \normalfont \Large \sffamily \bfseries
10289       \bool_if:NTF \l_@@_in_code_after_bool
10290         { \color { red } }
10291         { \color { red ! 50 } }
10292       ##1 - ####1
10293     }
10294     \bool_if:NTF \l_@@_in_code_after_bool
10295     {
10296       \pgfpicture
10297       \pgfrememberpicturerepositiononpagetrue
10298       \pgf@relevantforpicturesizefalse
10299     }
10300     { \begin { pgfpicture } }
10301     \@@_qpoint:n { col - ####1 }
10302     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10303     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10304     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10305     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10306     \bool_if:NTF \l_@@_in_code_after_bool
10307       { \endpgfpicture }
10308       { \end { pgfpicture } }
10309     \fp_set:Nn \l_tmpa_fp
10310     {
10311       \fp_min:nn
10312       {
10313         \fp_min:nn
10314         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10315         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10316       }
10317       { 1.0 }
10318     }
10319     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10320     \pgfpicture
10321     \pgfrememberpicturerepositiononpagetrue
10322     \pgf@relevantforpicturesizefalse
10323     \pgftransformshift
10324     {
10325       \pgfpoint
10326       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10327       \g_tmpa_dim
10328     }
10329     \pgfnode
10330     { rectangle }
10331     { center }
10332     { \box_use:N \l_tmpa_box }
10333     { }
10334     { }
10335     \endpgfpicture

```

```

10336     }
10337   }
10338 }

```

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.

```

10339 \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.

```

10340 \bool_new:N \g_@@_footnote_bool

10341 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10342 {
10343   You-have-used-the-key~' \l_keys_key_str '~when~loading~nicematrix~
10344   but~that~key~is~unknown. \\
10345   It~will~be~ignored. \\
10346   For~a~list~of~the~available~keys,~type~H~<return>.
10347 }
10348 {
10349   The~available~keys~are~(in~alphabetic~order):~
10350   footnote,~
10351   footnotehyper,~
10352   messages-for-Overleaf,~
10353   renew-dots~and~
10354   renew-matrix.
10355 }

10356 \keys_define:nn { nicematrix }
10357 {
10358   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10359   renew-dots .value_forbidden:n = true ,
10360   renew-matrix .code:n = \@@_renew_matrix: ,
10361   renew-matrix .value_forbidden:n = true ,
10362   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10363   footnote .bool_set:N = \g_@@_footnote_bool ,
10364   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10365   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10366 }

10367 \ProcessKeyOptions

10368 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10369 {
10370   You~can't~use~the~option~'footnote'~because~the~package~
10371   footnotehyper~has~already~been~loaded.~
10372   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10373   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10374   of~the~package~footnotehyper.\\
10375   The~package~footnote~won't~be~loaded.
10376 }

```

```

10377 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10378 {
10379   You~can't~use~the~option~'footnotehyper'~because~the~package~
10380   footnote~has~already~been~loaded.~
10381   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10382   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10383   of~the~package~footnote.\\
10384   The~package~footnotehyper~won't~be~loaded.
10385 }

```

```

10386 \bool_if:NT \g_@@_footnote_bool
10387 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10388   \IfClassLoadedTF { beamer }
10389   { \bool_set_false:N \g_@@_footnote_bool }
10390   {
10391     \IfPackageLoadedTF { footnotehyper }
10392     { \@@_error:n { footnote~with~footnotehyper~package } }
10393     { \usepackage { footnote } }
10394   }
10395 }

```

```

10396 \bool_if:NT \g_@@_footnotehyper_bool
10397 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10398   \IfClassLoadedTF { beamer }
10399   { \bool_set_false:N \g_@@_footnote_bool }
10400   {
10401     \IfPackageLoadedTF { footnote }
10402     { \@@_error:n { footnotehyper~with~footnote~package } }
10403     { \usepackage { footnotehyper } }
10404   }
10405   \bool_set_true:N \g_@@_footnote_bool
10406 }

```

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.

```

10407 \bool_new:N \l_@@_underscore_loaded_bool
10408 \IfPackageLoadedT { underscore }
10409 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10410 \AtBeginDocument
10411 {
10412   \bool_if:NF \l_@@_underscore_loaded_bool
10413   {
10414     \IfPackageLoadedT { underscore }
10415     { \@@_error:n { underscore~after~nicematrix } }
10416   }
10417 }

```

41 Compatibility with threeparttable

```

10418 \AtBeginDocument
10419 {
10420   \IfPackageLoadedT { threeparttable }
10421   {
10422     \AddToHook { env / threeparttable / begin }
10423     {
10424       \TPT@hookin { NiceTabular }
10425       \TPT@hookin { NiceTabular* }
10426       \TPT@hookin { NiceTabularX }
10427     }
10428   }
10429 }

```

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.

```

10430 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10431 {
10432   \str_if_eq:eeTF
10433   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10434   { + }
10435   {
10436     \str_set:Ne \l_tmpa_str
10437     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10438     \bool_set_false:N \l_tmpa_bool
10439     \clist_map_inline:nn { #1 }
10440     {
10441       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10442       {
10443         \@@_error:n { key~without~+~exists }
10444         \bool_set_true:N \l_tmpa_bool
10445         \clist_map_break:
10446       }
10447     }
10448     \bool_if:NF \l_tmpa_bool
10449     {
10450       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10451       \@@_unknown_key_i:nn { #1 } { #2 }
10452     }
10453   }
10454   { \@@_unknown_key_i:nn { #1 } { #2 } }
10455 }

```

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.

```

10456 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10457 {
10458   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10459   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10460   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10461   \bool_set_false:N \l_tmpa_bool

```



```

10462 \clist_map_inline:nn { #1 }
10463 {
10464     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10465     {
10466         \@@_error:n { key~with~normal~form~exists }
10467         \bool_set_true:N \l_tmpa_bool
10468         \clist_map_break:
10469     }
10470 }
10471 \bool_if:NF \l_tmpa_bool
10472 {
10473     \@@_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).

```

10474     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10475     { \msg_info:nn { nicematrix } { #2~+ } }
10476 }
10477 }
10478 \@@_msg_new:nn { key~without~+~exists }
10479 {
10480     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10481     additive~syntax~(with~+=)~.~It~will~be~ignored.
10482 }
10483 \@@_msg_new:nn { key~with~normal~form~exists }
10484 {
10485     No~key~'\l_keys_key_str'.\\
10486     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10487 }
10488 \str_const:Ne \c_@@_available_keys_str
10489 {
10490     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10491     { For~a~list~of~the~available~keys,~type~H~<return>. }
10492 }
10493 \seq_new:N \g_@@_types_of_matrix_seq
10494 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10495 {
10496     NiceMatrix ,
10497     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10498 }
10499 \seq_gset_map_e:Nn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10500 { \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`.

```

10501 \cs_new_protected:Npn \@@_err_too_many_cols:
10502 {
10503     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10504     { \@@_fatal:nn { too-many-cols-for-array } }
10505     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10506     { \@@_fatal:n { too-many-cols-for-matrix } }
10507     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10508     { \@@_fatal:n { too-many-cols-for-matrix } }
10509     \bool_if:NF \l_@@_last_col_without_value_bool
10510     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10511 }

```

The following command must *not* be protected since it's used in an error message.

```

10512 \cs_new:Npn \@@_message_hdotsfor:
10513 {
10514   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10515   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10516     \token_to_str:N \Hbrace \ is~incorrect. }
10517 }

10518 \cs_new_protected:Npn \@@_Hline_in_cell:
10519 { \@@_error:n { Misuse~of~Hline } }

10520 \@@_msg_new:nn { Misuse~of~Hline }
10521 {
10522   Misuse~of~\token_to_str:N \Hline. \\
10523   Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:.~
10524   \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must~be~used~only~
10525   at~the~beginning~of~a~row.~Your~command~will~be~ignored.
10526 }

10527 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10528 {
10529   Incompatible~options.\\
10530   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.~
10531   The~output~will~not~be~reliable.
10532 }

10533 \@@_msg_new:nn { Body~alone }
10534 {
10535   \token_to_str:N \Body\ alone. \\
10536   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10537 }

10538 \@@_msg_new:nn { Bad~use~of~CodeBefore }
10539 {
10540   Bad~use~of~\token_to_str:N \CodeBefore. \\
10541   \token_to_str:N \CodeBefore\ must~be~used~only~at~the~beginning~of~
10542   the~environment.
10543 }

10544 \@@_msg_new:nn { NiceTabularX~probably~required }
10545 {
10546   Incorrect~syntax.\\
10547   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10548   ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10549   with~the~key~'width'.
10550 }

10551 \@@_msg_new:nn { cellcolor~in~Block }
10552 {
10553   Bad~use~of~\token_to_str:N \cellcolor \\
10554   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10555   \bool_if:NTF \l_@@_amp_in_blocks_bool
10556   { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10557   { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10558   .~Here,~you~should~use~the~key~'fill'~of~the~block.~Your~command~will~be~ignored.
10559 }

10560 \@@_msg_new:nn { rowcolor~in~Block }
10561 {
10562   Bad~use~of~\token_to_str:N \rowcolor \\
10563   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10564   However,~it's~possible~to~color~the~block~with~its~key~'fill'.~
10565   Your~command~will~be~ignored.
10566 }

10567 \@@_msg_new:nn { key~color~inside }
10568 {
10569   Deleted~key.\\

```

```

10570     The-key~'color-inside'~(and-its-alias~'colortbl-like')~has-been-deleted-in
10571     ~'nicematrix'~and-must-not-be-used.
10572 }

10573 \@@_msg_new:nn { Invalid~argument~for~w }
10574 {
10575     Invalid~argument~for~w.\\
10576     You-have-used-the-type-of-alignment~'#1'~for-your-column~'w'~
10577     but-only~'c',~'r',~'l'~and~'s'~are-allowed-in-a-column~'w'.~
10578     If-you-go-on,~'c'~will-be-used.
10579 }

10580 \@@_msg_new:nn { invalid~weight }
10581 {
10582     Unknown~key.\\
10583     The-key~'\l_keys_key_str'~of-your-column~X~is-unknown-and-will-be-ignored.~
10584     The-available-keys-are:~l,~c,~r,~t~(=p),~m,~b,~V~
10585     \IfPackageLoadedTF { varwidth }
10586     { (since~'varwidth'~is-loaded)~}
10587     { (if~you~load~'varwidth')~}
10588     and-real-numbers-for-the-weight-of-the~X~column.
10589 }

10590 \@@_msg_new:nn { last~col~not~used }
10591 {
10592     Column~not~used.\\
10593     The-key~'last-col'~is-in-force-but-you-have-not-used-that~last~column~
10594     in-your~\@@_full_name_env: .~However,~you-can-go-on.
10595 }

10596 \@@_msg_new:nn { too-many~cols~for~matrix~with~last~col }
10597 {
10598     Too-many~columns.\\
10599     In-the~row~ \int_eval:n { \c@iRow },~
10600     you-try-to-use-more-columns~
10601     than-allowed-by-your~ \@@_full_name_env: .
10602     \@@_message_hdotsfor: \
10603     The-maximal-number-of-columns-is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10604     (plus-the-exterior-columns).~
10605     But,~maybe,~you-have-forgotten-a~\token_to_str:N \\.
10606 }

10607 \@@_msg_new:nn { too-many~cols~for~matrix }
10608 {
10609     Too-many~columns.\\
10610     In-the~row~ \int_eval:n { \c@iRow },~
10611     you-try-to-use-more-columns-than-allowed-by-your~ \@@_full_name_env: .
10612     \@@_message_hdotsfor: \
10613     Recall~that~the-maximal-number-of-columns-for-a-matrix~
10614     (excepted-the-potential-exterior-columns)~is-fixed-by-the~
10615     LaTeX-counter~'MaxMatrixCols'.~
10616     Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
10617     (use~ \token_to_str:N \setcounter \ to-change-that-value).~
10618     But,~maybe,~you-have-forgotten-a~\token_to_str:N \\.
10619 }

10620 \@@_msg_new:nn { too-many~cols~for~array }
10621 {
10622     Too-many~columns.\\
10623     In-the~row~ \int_eval:n { \c@iRow },~
10624     ~you-try-to-use-more-columns-than-allowed-by-your~
10625     \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
10626     \int_use:N \g_@@_static_num_of_col_int \
10627     \bool_if:nT
10628     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10629     { (plus-the-exterior-ones)~}

```

```

10630     since~the~preamble~is~' \g_@@_user_preamble_tl ' .~
10631     But,~maybe,~you~have~forgotten~a~\token_to_str:N \\.
10632 }

10633 \@@_msg_new:nn { columns~not~used }
10634 {
10635     Columns~not~used.\\
10636     The~preamble~of~your~ \@@_full_name_env: \ is~
10637     '\exp_not:V \g_@@_user_preamble_tl'.~
10638     It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10639     columns~but~you~only~used~ \int_use:N \c@jCol .~The~columns~you~did~not~
10640     used~won't~be~created.~You~won't~have~similar~warning~till~the~end~of~the~document.
10641 }

10642 }

10643 \@@_msg_new:nn { Bad~use~of~NiceTabularNotes }
10644 {
10645     Bad~use~of~\token_to_str:N \NiceTabularNotes \\
10646     \token_to_str:N~\NiceTabularNotes\ should~be~used~only~after~at~tabular~
10647     which~uses~`notes/no-print`.~ Your~command~will~be~ignored.
10648 }

10649 \@@_msg_new:nn { empty~preamble }
10650 {
10651     Empty~preamble.\\
10652     The~preamble~of~your~ \@@_full_name_env: \ is~empty.
10653 }

10654 \@@_msg_new:nn { in~first~col }
10655 {
10656     Erroneous~use.\\
10657     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.~
10658     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-col'.
10659 }

10660 \@@_msg_new:nn { in~last~col }
10661 {
10662     Erroneous~use.\\
10663     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.~
10664     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-col'.
10665 }

10666 \@@_msg_new:nn { in~first~row }
10667 {
10668     Erroneous~use.\\
10669     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
10670     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-row'.
10671 }

10672 \@@_msg_new:nn { in~last~row }
10673 {
10674     Erroneous~use.\\
10675     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
10676     Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-row'.
10677 }

10678 \@@_msg_new:nn { TopRule~without~booktabs }
10679 {
10680     Erroneous~use.\\
10681     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
10682     You~should~load~'booktabs'~(before~or~after~'nicematrix').~
10683     Your~command~will~be~ignored.
10684 }

10685 \@@_msg_new:nn{ rotate~in~p~col }
10686 {
10687     \token_to_str:N \rotate\ forbidden.\\
10688     You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10689     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~

```

```

10690     If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
10691     consider~the~classical~command~\token_to_str:N \rotatebox.
10692 }
10693 \@@_msg_new:nn { TopRule~without~tikz }
10694 {
10695     Erroneous~use.\\
10696     You~can't~use~the~command~#1~because~TikZ~is~not~loaded.~
10697     You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
10698     \IfPackageLoadedF { booktabs }
10699         { You~should~also~load~'booktabs'~
10700           with~\token_to_str:N \usepackage \{booktabs\}.~ }
10701     Your~command~will~be~ignored.
10702 }
10703 \@@_msg_new:nn { caption~outside~float }
10704 {
10705     Key~caption~forbidden.\\
10706     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10707     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
10708 }
10709 \@@_msg_new:nn { short~caption~without~caption }
10710 {
10711     You~should~not~use~the~key~'short~caption'~without~'caption'.~
10712     However,~your~'short~caption'~will~be~used~as~'caption'.
10713 }
10714 \@@_msg_new:nn { double~closing~delimiter }
10715 {
10716     Double~delimiter.\\
10717     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10718     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
10719     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10720 }
10721 \@@_msg_new:nn { delimiter~after~opening }
10722 {
10723     Double~delimiter.\\
10724     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10725     delimiter.~That~delimiter~will~be~ignored.
10726 }
10727 \@@_msg_new:nn { bad~option~for~line~style }
10728 {
10729     Bad~line~style.\\
10730     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10731     is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
10732     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
10733 }
10734 \@@_msg_new:nn { draw~trees~with~no~cell~nodes }
10735 {
10736     Incompatible~keys.\\
10737     You~can't~use~the~key~'draw~trees~in~col'~here~because~the~key~'no~cell~nodes'~
10738     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10739     is~to~speed~compilation~up).~If~you~go~on,~that~key~will~be~ignored.
10740 }
10741 \@@_msg_new:nn { corners~with~no~cell~nodes }
10742 {
10743     Incompatible~keys.\\
10744     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10745     is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10746     is~to~speed~compilation~up).\\
10747     If~you~go~on,~that~key~will~be~ignored.
10748 }
10749 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }

```

```

10750 {
10751   Incompatible~keys.\\
10752   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10753   is~in~force~(you~should~deactive~the~key~'no~cell~nodes'~whose~only~goal~
10754   is~to~speed~compilation~up).~If~you~go~on,~those~extra~nodes~won't~be~created.
10755 }
10756 \@@_msg_new:nn { Identical~notes~in~caption }
10757 {
10758   Identical~tabular~notes.\\
10759   You~can't~put~several~notes~with~the~same~content~in~
10760   \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
10761   If~you~go~on,~the~output~will~probably~be~erroneous.
10762 }
10763 \@@_msg_new:nn { tabularnote~below~the~tabular }
10764 {
10765   \token_to_str:N \tabularnote \ forbidden\\
10766   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10767   of~your~tabular~because~the~caption~will~be~composed~below~
10768   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10769   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
10770   Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
10771   no~similar~error~will~raised~in~this~document.
10772 }
10773 \@@_msg_new:nn { Unknown~key~for~rules }
10774 {
10775   Unknown~key.\\
10776   There~are~only~three~keys~available~here:~'width',~'color'~and~
10777   'fix~vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10778 }
10779 \@@_msg_new:nn { Unknown~key~for~trees }
10780 {
10781   Unknown~key.\\
10782   There~are~only~three~keys~available~here:~width~color~and~
10783   rounded~corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
10784 }
10785 % \end{macrocode}
10786 %
10787 %
10788 % \begin{macrocode}
10789 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10790 {
10791   Unknown~key.\\
10792   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10793   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10794   and~ \token_to_str:N \Vbrace \ are:~'brace~shift(+)',~'color',~
10795   'horizontal~label(s)',~'shorten'~'shorten~end'~
10796   and~'shorten~start'.
10797 }
10798 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10799 {
10800   Unknown~key.\\
10801   There~is~only~two~keys~available~here:~
10802   'empty'~and~'not~empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10803 }
10804 \@@_msg_new:nn { Unknown~key~for~rotate }
10805 {
10806   Unknown~key.\\
10807   The~only~keys~available~here~are~'c'~and~'-90'.~
10808   Your~key~' \l_keys_key_str '~will~be~ignored.
10809 }
10810 \@@_msg_new:nnn { Unknown~key~for~custom~line }

```

```

10811 {
10812   Unknown~key.\\
10813   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'~
10814   It~you~go~on,~you~will~probably~have~other~errors. \\
10815   \c_@@_available_keys_str
10816 }
10817 {
10818   The~available~keys~are~(in~alphabetic~order):~
10819   ccommand,~
10820   color,~
10821   command,~
10822   dotted,~
10823   letter,~
10824   multiplicity,~
10825   sep~color,~
10826   tikz,~and~total~width.
10827 }
10828 \@@_msg_new:nnn { Unknown~key~for~default~line }
10829 {
10830   Unknown~key.\\
10831   The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'~
10832   It~you~go~on,~you~will~probably~have~other~errors. \\
10833   \c_@@_available_keys_str
10834 }
10835 {
10836   The~available~keys~are~(in~alphabetic~order):~
10837   color,~
10838   dotted,~
10839   multiplicity,~
10840   sep~color,~
10841   tikz,~and~total~width.
10842 }
10843 \@@_msg_new:nnn { Unknown~key~for~xdots }
10844 {
10845   Unknown~key.\\
10846   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10847   \c_@@_available_keys_str
10848 }
10849 {
10850   The~available~keys~are~(in~alphabetic~order):~
10851   'color',~
10852   'horizontal(s)-labels',~
10853   'inter',~
10854   'line-style',~
10855   'nullify',~
10856   'radius',~
10857   'shorten',~
10858   'shorten-end'~and~'shorten-start'.
10859 }
10860 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10861 {
10862   Unknown~key.\\
10863   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
10864   (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
10865 }
10866 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10867 {
10868   Error~with~'draw~trees~in~col' \\
10869   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10870   is~\int_use:N \c@jCol.~It~will~be~ignored.
10871 }
10872 \@@_msg_new:nn { Last~col~in~trees }

```

```

10873 {
10874   Error~with~'draw-trees-in-col' \\
10875   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
10876   the~last~column~of~your~tabular.~It~will~be~ignored.
10877 }
10878 \@@_msg_new:nn { label~without~caption }
10879 {
10880   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10881   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10882 }
10883 \@@_msg_new:nn { W~warning }
10884 {
10885   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10886   (row~ \int_use:N \c@iRow ).
10887 }
10888 \@@_msg_new:nn { Construct~too~large }
10889 {
10890   Construct~too~large.\\
10891   Your~command~ \token_to_str:N #1
10892   can't~be~drawn~because~your~matrix~is~too~small.~
10893   Your~command~will~be~ignored.
10894 }
10895 \@@_msg_new:nn { underscore~after~nicematrix }
10896 {
10897   Problem~with~'underscore'.\\
10898   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10899   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10900   ' \token_to_str:N \Cdots \token_to_str:N _
10901   \{ n \token_to_str:N \text \{ ~times \} \}' .
10902 }
10903 \@@_msg_new:nn { ampersand~in~light-syntax }
10904 {
10905   Ampersand~forbidden.\\
10906   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10907   the~key~'light-syntax'~is~in~force.
10908 }
10909 \@@_msg_new:nn { double~backslash~in~light-syntax }
10910 {
10911   Double~backslash~forbidden.\\
10912   You~can't~use~ \token_to_str:N \\
10913   ~to~separate~rows~because~the~key~'light-syntax'~
10914   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10915   (set~by~the~key~'end-of-row').
10916 }
10917 \@@_msg_new:nn { bad~value~for~baseline }
10918 {
10919   Bad~value~for~baseline.\\
10920   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10921   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10922   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10923   the~form~'line-i'.~A~value~of~1~will~be~used.
10924 }
10925 \@@_msg_new:nn { bad~value~for~baseline~line }
10926 {
10927   Bad~value~for~baseline~with~line.\\
10928   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10929   valid.~The~number~of~the~line~must~be~between~1~and~
10930   \int_eval:n { \c@iRow + 1 }.~
10931   A~value~of~'line-1'~will~be~used.
10932 }

```



```

10933 \@@_msg_new:nn { detection-of-empty-cells }
10934 {
10935   Problem-with~'not-empty'\
10936   For~technical~reasons,~you~must~activate~
10937   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10938   in~order~to~use~the~key~' \l_keys_key_str '~
10939   Your~key~will~be~ignored.
10940 }
10941 \@@_msg_new:nn { siunitx-too-old }
10942 {
10943   siunitx~too-old\
10944   You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
10945   is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26).
10946 }
10947 \@@_msg_new:nn { Invalid-name }
10948 {
10949   Invalid-name.\
10950   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10951   \SubMatrix \ of~your~ \@@_full_name_env: .~
10952   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
10953   Your~key~will~be~ignored.
10954 }
10955 \@@_msg_new:nn { Hbrace-not-allowed }
10956 {
10957   Command~not~allowed.\
10958   You~can't~use~the~command~ \token_to_str:N #1
10959   because~you~have~not~loaded~
10960   \IfPackageLoadedTF { tikz }
10961     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10962     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10963     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10964     Your~command~will~be~ignored.
10965 }
10966 \@@_msg_new:nn { Vbrace-not-allowed }
10967 {
10968   Command~not~allowed.\
10969   You~can't~use~the~command~ \token_to_str:N \Vbrace \
10970   because~you~have~not~loaded~TikZ~
10971   and~the~TikZ~library~'decorations.pathreplacing'~
10972   Use: ~\token_to_str:N \usepackage \{tikz\}~
10973   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10974   Your~command~will~be~ignored.
10975 }
10976 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10977 {
10978   Wrong~line.\
10979   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10980   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10981   number~is~not~valid.~It~will~be~ignored.
10982 }
10983 \@@_msg_new:nn { Impossible-SubMatrix }
10984 {
10985   Impossible~SubMatrix.\
10986   It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
10987   because~all~the~cells~are~empty~in~the~column~on~the~#1~
10988   side~of~that~\token_to_str:N \SubMatrix.
10989   \bool_if:NT \l_@@_submatrix_slim_bool
10990     { ~Maybe~you~should~try~without~the~key~'slim'. } \
10991   This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10992 }
10993 \@@_msg_new:nn { Impossible-SubMatrix-no-cell-nodes }

```

```

10994 {
10995     Impossible~SubMatrix.\\
10996     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
10997     because~the~key~'no-cell-nodes'~is~in~force.~
10998     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10999 }

11000 \@@_msg_new:nnn { width~without~X~columns }
11001 {
11002     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11003     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11004     Your~key~will~be~ignored.
11005 }
11006 {
11007     This~message~is~the~message~'width~without~X~columns'~
11008     of~the~module~'nicematrix'.~
11009     The~experimented~users~can~disable~that~message~with~
11010     \token_to_str:N \msg_redirect_name:nnn .\\
11011 }
11012

11013 \@@_msg_new:nn { key~multiplicity~with~dotted }
11014 {
11015     Incompatible~keys. \\
11016     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11017     in~a~'custom-line'.~They~are~incompatible.~
11018     The~key~'multiplicity'~will~be~ignored.
11019 }

11020 \@@_msg_new:nn { empty~environment }
11021 {
11022     Empty~environment.\\
11023     Your~ \@@_full_name_env: \ is~empty.
11024 }

11025 \@@_msg_new:nn { No~letter~and~no~command }
11026 {
11027     Erroneous~use.\\
11028     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
11029     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11030     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11031 }

11032 \@@_msg_new:nn { Forbidden~letter }
11033 {
11034     Forbidden~letter.\\
11035     You~can't~use~the~letter~'#1'~for~a~customized~line.~
11036     It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11037 }

11038 \@@_msg_new:nn { Several~letters }
11039 {
11040     Wrong~name.\\
11041     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11042     have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11043 }

11044 \@@_msg_new:nn { Delimiter~with~small }
11045 {
11046     Delimiter~forbidden.\\
11047     You~can't~put~a~delimiter~in~the~preamble~of~your~
11048     \@@_full_name_env: \
11049     because~the~key~'small'~is~in~force.
11050 }

11051 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11052 {
11053     Unknown~cell.\\

```

```

11054 Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11055 the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11056 can't~be~executed~because~a~cell~doesn't~exist.~
11057 Your~command~ \token_to_str:N \line \ will~be~ignored.
11058 }
11059 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11060 {
11061   Duplicate~name. \\
11062   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11063   in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11064   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11065   { For~a~list~of~the~names~already~used,~type~H~<return>. }
11066 }
11067 {
11068   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11069   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11070 }
11071 \@@_msg_new:nn { r~or~l~with~preamble }
11072 {
11073   Erroneous~use. \\
11074   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11075   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11076   your~ \@@_full_name_env: .~
11077   Your~key~will~be~ignored.
11078 }
11079 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11080 {
11081   Erroneous~use. \\
11082   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11083   in~an~exterior~column~of~the~array.
11084 }
11085 \@@_msg_new:nn { bad~corner }
11086 {
11087   Bad~corner. \\
11088   '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11089   'corners')~.~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11090   This~specification~of~corner~will~be~ignored.
11091 }
11092 \@@_msg_new:nn { bad~border }
11093 {
11094   Bad~border.\\
11095   '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11096   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11097   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11098   also~use~the~key~'tikz'
11099   \IfPackageLoadedF { tikz }
11100   { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11101   This~specification~of~border~will~be~ignored.
11102 }
11103 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11104 {
11105   TikZ~not~loaded. \\
11106   You~can't~use~ \token_to_str:N \TikzEveryCell \
11107   because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11108   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11109   Your~command~will~be~ignored.
11110 }
11111 \@@_msg_new:nn { tikz~key~without~tikz }
11112 {
11113   TikZ~not~loaded. \\
11114   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N

```

```

11115 \Block '~because-you-have-not-loaded-TikZ.~
11116 You-can-load-TikZ-with-\token_to_str:N \usepackage \{tikz\},~
11117 before-or-after-'nicematrix'.~Your-key-will-be-ignored.
11118 }
11119 \@@_msg_new:nn { Bad~argument~for~Block }
11120 {
11121   Bad~argument. \\
11122   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11123   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11124   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
11125   the~argument~was~empty).
11126 }
11127 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11128 {
11129   Erroneous~use. \\
11130   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11131   'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11132   (the~value~' \l_keys_value_tl '~will~be~ignored).
11133 }
11134 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11135 {
11136   Erroneous~use. \\
11137   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11138   'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11139   (the~value~' \l_keys_value_tl '~will~be~ignored).
11140 }
11141 \@@_msg_new:nn { Block~too~large~1 }
11142 {
11143   Block~too~large. \\
11144   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
11145   too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11146 }
11147 \@@_msg_new:nn { Block~too~large~2 }
11148 {
11149   Block~too~large. \\
11150   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
11151   \g_@@_static_num_of_col_int \
11152   columns~but~you~use~only~ \int_use:N \c_jCol \ and~that's~why~a~block~
11153   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
11154   (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: .~
11155   This~block~and~maybe~others~will~be~ignored.
11156 }
11157 \@@_msg_new:nn { unknown~column~type }
11158 {
11159   Bad~column~type. \\
11160   The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.
11161 }
11162 \@@_msg_new:nn { unknown~column~type~multicolumn }
11163 {
11164   Bad~column~type. \\
11165   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
11166   ~of~your~ \@@_full_name_env: \ is~unknown.
11167 }
11168 \@@_msg_new:nn { unknown~column~type~S }
11169 {
11170   Bad~column~type. \\
11171   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown.~
11172   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11173   load~that~package~with~\token_to_str:N \usepackage \{siunitx\}.
11174 }

```

```

11175 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
11176 {
11177   Bad~column~type. \\
11178   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11179   of~your~ \@@_full_name_env: \ is~unknown.~If~you~want~to~use~the~column~
11180   type~'S'~of~siunitx,~you~should~load~that~package~with~
11181   \token_to_str:N \usepackage \{siunitx\}.
11182 }
11183 \@@_msg_new:nn { tabularnote~forbidden }
11184 {
11185   Forbidden~command. \\
11186   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11187   ~here.~This~command~is~available~only~in~
11188   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11189   the~argument~of~a~command~\token_to_str:N \caption \ included~
11190   in~an~environment~\{table\}.~Your~command~will~be~ignored.
11191 }
11192 \@@_msg_new:nn { borders~forbidden }
11193 {
11194   Forbidden~key.\\
11195   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11196   because~the~option~'rounded~corners'~is~in~force~with~a~non~zero~value.~
11197   Your~key~will~be~ignored.
11198 }
11199 \@@_msg_new:nn { bottomrule~without~booktabs }
11200 {
11201   booktabs~not~loaded.\\
11202   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11203   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11204   after~'nicematrix'.~Your~key~will~be~ignored.
11205 }
11206 \@@_msg_new:nn { enumitem~not~loaded }
11207 {
11208   enumitem~not~loaded. \\
11209   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11210   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11211   (before~or~after~'nicematrix').~All~the~commands~
11212   \token_to_str:N \tabularnote \ will~be~ignored~in~the~document.
11213 }
11214 \@@_msg_new:nn { tikz~without~tikz }
11215 {
11216   TikZ~not~loaded. \\
11217   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11218   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11219   If~you~go~on,~your~key~will~be~ignored.
11220 }
11221 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11222 {
11223   TikZ~not~loaded. \\
11224   You~have~used~the~key~'tikz'~in~the~definition~of~a~
11225   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
11226   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11227   use~that~custom~line.~You~should~load~TikZ~with~
11228   \token_to_str:N \usepackage \{tikz\}.
11229 }
11230 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11231 {
11232   TikZ~not~loaded. \\
11233   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11234   command~ \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11235   Your~key~will~be~ignored.~You~should~load~TikZ~with~

```

```

11236 \token_to_str:N \usepackage \{tikz\}
11237 }
11238 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11239 {
11240   Erroneous~use.\
11241   In~a~'custom~line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11242   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11243   The~key~'color'~will~be~ignored.
11244 }
11245 \@@_msg_new:nn { Wrong~last~row }
11246 {
11247   Wrong~number.\
11248   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
11249   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11250   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11251   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11252   problem~by~using~'last~row'~without~value~(more~compilations~
11253   might~be~necessary).
11254 }
11255 \@@_msg_new:nn { Yet~in~env }
11256 {
11257   Nested~environments.\
11258   Environments~of~nicematrix~can't~be~nested.~However~you~
11259   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11260   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11261   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11262   that~box~in~another~environment~of~nicematrix.
11263 }
11264 \@@_msg_new:nn { Outside~math~mode }
11265 {
11266   Outside~math~mode.\
11267   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11268   (and~not~in~ \token_to_str:N \vcenter ).
11269 }
11270 \@@_msg_new:nn { One~letter~allowed }
11271 {
11272   Bad~name.\
11273   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11274   you~have~used~' \l_keys_value_tl '.~Your~key~will~be~ignored.
11275 }
11276 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11277 {
11278   Environment~\{TabularNote\}~forbidden.\
11279   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11280   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11281   \{TabularNote\}~will~be~ignored.
11282 }
11283 \@@_msg_new:nn { varwidth~not~loaded }
11284 {
11285   varwidth~not~loaded.\
11286   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11287   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11288   Your~column~will~behave~like~'p'.
11289 }
11290 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11291 {
11292   varwidth~not~loaded.\
11293   You~can't~use~the~key~'V'~in~your~column~'X'~
11294   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11295   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11296 }

```

```

11297 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11298 {
11299   Unknown~key.\
11300   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\
11301   \c_@@_available_keys_str
11302 }
11303 {
11304   The~available~keys~are:~color,~multiplicity,~sep-color,~tikz~
11305   and~total-width~(the~latter~is~meaningful~only~in~cunjunction~with~'tikz').
11306 }

```

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `:`.

```

11307 \@@_msg_new:nnn { Unknown~key~for~Block }
11308 {
11309   Unknown~key. \
11310   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11311   \token_to_str:N \Block .~Your~key~will~be~ignored. \
11312   \c_@@_available_keys_str
11313 }
11314 {
11315   The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
11316   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11317   opacity,~rounded-corners,~r,~respect-arraystretch,~rules/width,~t,~T,~tikz,~
11318   transparent~and~vlines.
11319 }
11320 \@@_msg_new:nnn { Unknown~key~for~Brace }
11321 {
11322   Unknown~key.\
11323   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11324   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11325   Your~key~will~be~ignored. \
11326   \c_@@_available_keys_str
11327 }
11328 {
11329   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
11330   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
11331   right-shorten)~and~yshift.
11332 }
11333 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11334 {
11335   Unknown~key.\
11336   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \
11337   \c_@@_available_keys_str
11338 }
11339 {
11340   The~available~keys~are~(in~alphabetic~order):~
11341   delimiters/color,~
11342   rules~(with~the~subkeys~'color'~and~'width'),~
11343   sub-matrix~(several~subkeys)~
11344   and~xdots~(several~subkeys).~
11345   The~latter~is~for~the~command~ \token_to_str:N \line .
11346 }
11347 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11348 {
11349   Unknown~key.\
11350   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \
11351   \c_@@_available_keys_str
11352 }
11353 {
11354   The~available~keys~are~(in~alphabetic~order):~
11355   create-cell-nodes,~

```

```

11356     delimiters/color~and~
11357     sub-matrix~(several~subkeys).
11358 }

11359 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11360 {
11361     Unknown~key.\\
11362     The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11363     \c_@@_available_keys_str
11364 }
11365 {
11366     The~available~keys~are~(in~alphabetic~order):~
11367     'delimiters/color',~
11368     'extra-height',~
11369     'hlines',~
11370     'hvlines',~
11371     'left-xshift',~
11372     'name',~
11373     'right-xshift',~
11374     'rules'~(with~the~subkeys~'color'~and~'width'),~
11375     'slim',~
11376     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11377 }

11378 \@@_msg_new:nnn { Unknown~key~for~notes }
11379 {
11380     Unknown~key.\\
11381     The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11382     \c_@@_available_keys_str
11383 }
11384 {
11385     The~available~keys~are~(in~alphabetic~order):~
11386     bottomrule,~
11387     code-after,~
11388     code-before(+),~
11389     detect-duplicates,~
11390     enumitem-keys,~
11391     enumitem-keys-para,~
11392     para,~
11393     label-in-list,~
11394     label-in-tabular~and~
11395     style.
11396 }

11397 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11398 {
11399     Unknown~key.\\
11400     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11401     \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11402     \c_@@_available_keys_str
11403 }
11404 {
11405     The~available~keys~are~(in~alphabetic~order):~
11406     bold,~
11407     cell-space-top-limit(+),~
11408     cell-space-bottom-limit(+),~
11409     cell-space-limits(+),~
11410     color,~
11411     fill~(alias:~rowcolor),~
11412     nb-rows,~
11413     opacity~and~
11414     rounded-corners.
11415 }

11416 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11417 {

```



```

11418   Unknown~key.\\
11419   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11420   \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11421   \c_@@_available_keys_str
11422 }
11423 {
11424   The~available~keys~are~(in~alphabetic~order):~
11425   &~in~blocks,~
11426   allow~duplicate~names,~
11427   ampersand~in~blocks,~
11428   caption~above,~
11429   cell~space~bottom~limit(+),~
11430   cell~space~limits(+),~
11431   cell~space~top~limit(+),~
11432   code~for~first~col(+),~
11433   code~for~first~row(+),~
11434   code~for~last~col(+),~
11435   code~for~last~row(+),~
11436   corners,~
11437   custom~key,~
11438   create~extra~nodes,~
11439   create~medium~nodes,~
11440   create~large~nodes,~
11441   custom~line,~
11442   delimiters~(several~subkeys),~
11443   end~of~row,~
11444   first~col,~
11445   first~row,~
11446   h(v)lines,~
11447   h(v)lines~except~borders,~
11448   last~col,~
11449   last~row,~
11450   left~margin,~
11451   light~syntax,~
11452   light~syntax~expanded,~
11453   matrix/columns~type,~
11454   no~cell~nodes,~
11455   notes~(several~subkeys),~
11456   nullify~dots,~
11457   pgf~node~code,~
11458   renew~dots,~
11459   renew~matrix,~
11460   respect~arraystretch,~
11461   rounded~corners,~
11462   right~margin,~
11463   rules~(with~the~subkeys~'color'~and~'width'),~
11464   small,~
11465   sub~matrix~(several~subkeys),~
11466   vlines,~
11467   xdots~(several~subkeys).
11468 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11469 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11470 {
11471   Unknown~key.\\
11472   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11473   \{NiceArray\}.~Your~key~will~be~ignored. \\
11474   \c_@@_available_keys_str
11475 }
11476 {
11477   The~available~keys~are~(in~alphabetic~order):~
11478   &~in~blocks,~

```

```

11479     ampersand-in-blocks,~
11480     b,~
11481     baseline,~
11482     c,~
11483     cell-space-bottom-limit,~
11484     cell-space-limits,~
11485     cell-space-top-limit,~
11486     code-after,~
11487     code-for-first-col(+),~
11488     code-for-first-row(+),~
11489     code-for-last-col(+),~
11490     code-for-last-row(+),~
11491     columns-width,~
11492     corners,~
11493     create-blocks-in-col,~
11494     create-extra-nodes,~
11495     create-medium-nodes,~
11496     create-large-nodes,~
11497     draw-trees-in-col,~
11498     extra-left-margin,~
11499     extra-right-margin,~
11500     first-col,~
11501     first-row,~
11502     h(v)lines,~
11503     h(v)lines-except-borders,~
11504     last-col,~
11505     last-row,~
11506     left-margin,~
11507     light-syntax,~
11508     light-syntax-expanded,~
11509     name,~
11510     no-cell-nodes,~
11511     nullify-dots,~
11512     pgf-node-code,~
11513     renew-dots,~
11514     respect-arraystretch,~
11515     right-margin,~
11516     rounded-corners,~
11517     rules~(with~the~subkeys~'color'~and~'width'),~
11518     small,~
11519     t,~
11520     vlines,~
11521     xdots/color,~
11522     xdots/shorten-start(+),~
11523     xdots/shorten-end(+),~
11524     xdots/shorten(+)-and~
11525     xdots/line-style.
11526 }

```

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`).

```

11527 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11528 {
11529     Unknown~key.\\
11530     The~key~' \l_keys_key_str '~is~unknown~for~the~
11531     \@@_full_name_env: .~Your~key~will~be~ignored. \\
11532     \c_@@_available_keys_str
11533 }
11534 {
11535     The~available~keys~are~(in~alphabetic~order):~
11536     &-in-blocks,~
11537     ampersand-in-blocks,~
11538     b,~

```

```

11539 baseline,~
11540 c,~
11541 cell-space-bottom-limit,~
11542 cell-space-limits,~
11543 cell-space-top-limit,~
11544 code-after,~
11545 code-for-first-col(+),~
11546 code-for-first-row(+),~
11547 code-for-last-col(+),~
11548 code-for-last-row(+),~
11549 columns-type,~
11550 columns-width,~
11551 corners,~
11552 create-blocks-in-col,~
11553 create-extra-nodes,~
11554 create-medium-nodes,~
11555 create-large-nodes,~
11556 draw-trees-in-col,~
11557 extra-left-margin,~
11558 extra-right-margin,~
11559 first-col,~
11560 first-row,~
11561 h(v)lines,~
11562 h(v)lines-except-borders,~
11563 l,~
11564 last-col,~
11565 last-row,~
11566 left-margin,~
11567 light-syntax,~
11568 light-syntax-expanded,~
11569 name,~
11570 no-cell-nodes,~
11571 nullify-dots,~
11572 pgf-node-code,~
11573 r,~
11574 renew-dots,~
11575 respect-arraystretch,~
11576 right-margin,~
11577 rounded-corners,~
11578 rules~(with~the~subkeys~'color'~and~'width'),~
11579 small,~
11580 t,~
11581 vl_lines,~
11582 xdots/color,~
11583 xdots/shorten-start(+),~
11584 xdots/shorten-end(+),~
11585 xdots/shorten(+),~and~
11586 xdots/line-style.
11587 }

11588 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11589 {
11590   Unknown~key.\\
11591   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11592   \{NiceTabular\}.~Your~key~will~be~ignored. \\
11593   \c_@@_available_keys_str
11594 }
11595 {
11596   The~available~keys~are~(in~alphabetic~order):~
11597   &~in~blocks,~
11598   ampersand~in~blocks,~
11599   b,~
11600   baseline,~
11601   c,~

```

```

11602 caption,~
11603 cell-space-bottom-limit,~
11604 cell-space-limits,~
11605 cell-space-top-limit,~
11606 code-after,~
11607 code-for-first-col(+),~
11608 code-for-first-row(+),~
11609 code-for-last-col(+),~
11610 code-for-last-row(+),~
11611 columns-width,~
11612 corners,~
11613 custom-line,~
11614 create-blocks-in-col,~
11615 create-extra-nodes,~
11616 create-medium-nodes,~
11617 create-large-nodes,~
11618 draw-trees-in-col,~
11619 extra-left-margin,~
11620 extra-right-margin,~
11621 first-col,~
11622 first-row,~
11623 h(v)lines,~
11624 h(v)lines-except-borders,~
11625 label,~
11626 last-col,~
11627 last-row,~
11628 left-margin,~
11629 light-syntax,~
11630 light-syntax-expanded,~
11631 name,~
11632 no-cell-nodes,~
11633 notes~(several~subkeys),~
11634 nullify-dots,~
11635 pgf-node-code,~
11636 renew-dots,~
11637 respect-arraystretch,~
11638 right-margin,~
11639 rounded-corners,~
11640 rules~(with~the~subkeys~'color'~and~'width'),~
11641 short-caption,~
11642 t,~
11643 tabularnote,~
11644 vlines,~
11645 xdots/color,~
11646 xdots/shorten-start(+),~
11647 xdots/shorten-end(+),~
11648 xdots/shorten(+),~and~
11649 xdots/line-style.
11650 }

11651 \@@_msg_new:nnn { Duplicate-name }
11652 {
11653   Duplicate-name.\
11654   The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
11655   the-same-environment-name-twice.~You-can-go-on,~but,~
11656   maybe,~you-will-have-incorrect-results-especially~
11657   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
11658   message-again,~use-the-key~'allow-duplicate-names'~in~
11659   ' \token_to_str:N \NiceMatrixOptions '\
11660   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11661   { For-a-list-of-the-names-already-used,~type-H~<return>. }
11662 }
11663 {
11664   The-names-already-defined-in-this-document-are:~

```

```

11665 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11666 }
11667 \@@_msg_new:nn { caption-above-in-env }
11668 {
11669   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~
11670   Your-key-will-be-ignored.
11671 }
11672 \@@_msg_new:nn { show-cell-names }
11673 {
11674   There-is-no-key~'show-cell-names'~in~nicematrix.\\
11675   You-should-use-the-command~\token_to_str:N \ShowCellNames\
11676   in-the~\token_to_str:N \CodeBefore\ or-the~\token_to_str:N
11677   \CodeAfter.~Your-key-will-be-ignored.
11678 }
11679 \@@_msg_new:nn { Option-auto-for-columns-width }
11680 {
11681   Erroneous-use.\\
11682   You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
11683   Your-key-will-be-ignored.
11684 }
11685 \@@_msg_new:nn { NiceTabularX-without-X }
11686 {
11687   NiceTabularX-without-X.\\
11688   You-should-not-use~\{NiceTabularX\}~without-X-columns.~However,~you-can-go-on.
11689 }
11690 \@@_msg_new:nn { Preamble-forgotten }
11691 {
11692   Preamble-forgotten.\\
11693   You-have-probably-forgotten-the-preamble-of-your~
11694   \@@_full_name_env: .
11695 }
11696 \@@_msg_new:nn { Invalid-col-number }
11697 {
11698   Invalid-column-number.\\
11699   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
11700   specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.~
11701   Maybe-this-is-a-spurious-error-due-to-an-incorrect~'aux'~file.
11702 }
11703 \@@_msg_new:nn { Invalid-row-number }
11704 {
11705   Invalid-row-number.\\
11706   A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
11707   specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.~
11708   Maybe-this-is-a-spurious-error-due-to-an-incorrect~'aux'~file.
11709 }
11710 \@@_define_com:NNN p ( )
11711 \@@_define_com:NNN b [ ]
11712 \@@_define_com:NNN v | |
11713 \@@_define_com:NNN V \l \l
11714 \@@_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>	37
9	The <code>\CodeBefore</code>	53
10	The environment <code>{NiceArrayWithDelims}</code>	57
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	80
13	The environment <code>{NiceMatrix}</code> and its variants	97
	13.1 Definition of <code>{pNiceMatrix}</code>	97
	13.2 The key <code>renew-matrix</code>	98
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	98
15	After the construction of the array	99
16	We draw the dotted lines	106
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	124
18	User commands available in the new environments	129
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	136
20	The command <code>\RowStyle</code>	137
21	Colors of cells, rows and columns	140
22	The vertical and horizontal rules	153
23	The empty corners	175
24	The environment <code>{NiceMatrixBlock}</code>	178
25	The extra nodes	179
26	The blocks	184
27	Automatic arrays	212
28	The redefinition of the command <code>\dotfill</code>	213
29	The command <code>\diagbox</code>	214

30	The keyword <code>\CodeAfter</code>	215
31	The delimiters in the preamble	216
32	The command <code>\SubMatrix</code>	217
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	226
34	The commands <code>HBrace</code> et <code>VBrace</code>	229
35	The command <code>TikzEveryCell</code>	232
36	The key <code>draw-trees-in-col</code>	234
37	The key <code>create-blocks-in-col</code>	235
38	The command <code>\ShowCellNames</code>	236
39	We process the options at package loading	238
40	About the package underscore	239
41	Compatibility with <code>threeparttable</code>	240
42	Error messages of the package	240