

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 7, 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~2026-06-01. \\
12   If~you~use~Overleaf,~you~need~at~least~"TeXLive~2026".\\
13   The~package~'nicematrix'~won't~be~loaded.
14 }
15 \IfFormatAtLeastTF { 2026-06-01 }
16 { }
17 { \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.

```
18 \RequirePackage { amsmath }
```

*This document corresponds to the version 7.12 of `nicematrix`, at the date of 2026/10/07.

```

19 \msg_new:nnn { nicematrix } { array-too-old }
20 {
21   Your~version-of~'array'~is~too~old. \
22   You~need~at~least~the~version~of~2025-09-25~and~the~version~that~
23   you~have~loaded~has~the~date:~
24   \str_range:cnn { ver @ array . sty } { 1 } { 10 }.~
25   The~package~'nicematrix'~won't~be~loaded.
26 }
27 \RequirePackage{array}
28 \IfPackageAtLeastF { array }
29 { 2025/09/25 }
30 { \msg_critical:nn { nicematrix } { array-too-old } }

31 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
32 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
33 \cs_new_protected:Npn \@@_warning:nnn { \msg_warning:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
35 \cs_generate_variant:Nn \@@_error:nn { n e }
36 \cs_new_protected:Npn \@@_error:nnnn { \msg_error:nnnn { nicematrix } }
37 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
38 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
39 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
40 \cs_new_protected:Npn \@@_critical:n { \msg_critical:nn { 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).

```

41 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
42 {
43   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
44   { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
45   {
46     \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).

```

47     \msg_new:nnn { nicematrix } { #1~+ } { #3 }
48   }
49 }

```

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

```

50 \cs_new_protected:Npn \@@_error_or_warning:n
51 {
52   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
53   \@@_warning:n
54   \@@_error:n
55 }

```

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".

```

56 \bool_new:N \g_@@_messages_for_Overleaf_bool
57 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
58 {
59   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
60   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
61 }

```

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`.

```

62 \@@_msg_new:nn { class~revtex }
63 {
64   nicematrix~is~incompatible~with~REVTeX\\
65   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
66   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...
67   The~package~'nicematrix'~won't~be~loaded.
68 }
69 \IfClassLoadedT { revtex4-1 } { \@@_critical:n { class~revtex } }
70 \IfClassLoadedT { revtex4-2 } { \@@_critical:n { class~revtex } }

```

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

```

71 \cs_if_exist:NT \rvtx@ifformat@geq { \@@_critical:n { class~revtex } }

72 \@@_msg_new:nn { mdwtab~loaded }
73 {
74   'nicematrix'~is~incompatible~with~'mdwtab'\\
75   This~error~is~fatal.
76 }

77 \AtBeginDocument
78 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab~loaded } } }

```

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`).

```

79 \cs_new_protected:Npn \@@_collect_options:n #1
80 {
81   \peek_meaning:NTF [
82     { \@@_collect_options:nw { #1 } }
83     { #1 { } }
84   }

```

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

```

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

```

3 Technical definitions

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

```

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

105 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
106 \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).

```

107 \tl_new:N \l_@@_argspec_tl

108 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
109 \cs_generate_variant:Nn \str_set:Nn { N o }
110 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
111 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
112 \cs_generate_variant:Nn \dim_min:nn { v }
113 \cs_generate_variant:Nn \dim_max:nn { v }

114 \AtBeginDocument
115 {
116   \IfPackageLoadedTF { tikz }
117   {

```

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.

```

118   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
119   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
120 }
121 {
122   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
123   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
124 }
125 }

```

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.

```

126 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
127 {
128   \iow_now:Nn \@mainaux
129   {
130     \ExplSyntaxOn
131     \cs_if_free:NT \pgfsyspdfmark
132     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
133     \ExplSyntaxOff

```

```

134     }
135     \cs_gset_eq:Nn \@@_provide_pgfsyspdfmark: \prg_do_nothing:
136 }

```

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.

```

137 \ProvideDocumentCommand { \iddots } { }
138 {
139     \mathinner
140     {
141         \mkern 1 mu
142         \box_move_up:nn { 1 pt } { \hbox { . } }
143         \mkern 2 mu
144         \box_move_up:nn { 4 pt } { \hbox { . } }
145         \mkern 2 mu
146         \box_move_up:nn { 7 pt }
147         { \vbox:n { \kern 7 pt \hbox { . } } }
148         \mkern 1 mu
149     }
150 }

```

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.

```

151 \AtBeginDocument
152 {
153     \IfPackageLoadedT { booktabs }
154     {
155         \iow_now:Nn \@mainaux
156         {
157             \ExplSyntaxOn
158             \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
159             \ExplSyntaxOff
160         }
161     }
162 }
163 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
164 {
165     \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`).

```

166     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
167     {
168         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
169         { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
170     }
171 }

```

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`.

```

172 \AtBeginDocument
173 {
174     \cs_set_protected:Npe \@@_everycr:
175     {
176         \IfPackageLoadedTF { colortbl } { \CT@everycr \everycr

```

```

177         { \noalign { \@@_in_everycr: } }
178     }
179     \IfPackageLoadedTF { colortbl }
180     {

```

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).

```

181         \cs_new_protected:Npn \@@_replace_columncolor:
182         {
183             \tl_replace_all:Nnn \l_@@_array_preamble_tl
184             \columncolor
185             \@@_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 color space.

```

186         }
187         \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
188         \cs_new_eq:NN \@@_old_rowcolor: \rowcolor

```

Since the final user may use a `{tabular}` within a cell of an environment of `nicematrix`, we have to revert the definitions of the commands `\cellcolor`, `\rowcolor` and `\multicolumn` at the beginning of such environments `{tabular}`.

```

189         \hook_gput_code:nnn { env / tabular / begin } { \nicematrix }
190         {
191             \bool_if:NT \l_@@_in_env_bool
192             {
193                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
194                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
195                 \cs_set_eq:NN \multicolumn \@@_old_multicolumn:
196             }
197         }
198     }
199 }

```

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.

```

200     \def \CT@arc@ { }
201     \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
202     \def \CT@arc #1 #2
203     {
204         \dim_compare:nNt \baselineskip = \c_zero_dim { \noalign {
205             { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
206         }

```

Idem for `\CT@drs@`.

```

207     \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
208     \def \CT@drs #1 #2
209     {
210         \dim_compare:nNt \baselineskip = \c_zero_dim { \noalign {
211             { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
212         }
213     }
214     \def \hline
215     {
216         \noalign { \ifnum 0 = ` } \fi
217         \cs_set_eq:NN \hskip \vskip
218         \cs_set_eq:NN \vrule \hrule
219         \cs_set_eq:NN \@width \@height
220         { \CT@arc@ \vline }
221         \futurelet \reserved@a
222         \@xhline
223     }
224     \cs_new_protected:Npn \@@_replace_columncolor:
225     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
226     \hook_gput_code:nnn { env / tabular / begin } { \nicematrix }

```

```

226     {
227         \bool_if:NT \l_@@_in_env_bool
228         { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
229     }
230 }
231 }
232 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

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.

```

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

```

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

```

243     \skip_horizontal:N \c_zero_dim
244 }

```

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`.

```

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

```

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.

```

249 \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`.

```

250 { \@@_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*.

```

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

```

¹See question 99041 on TeX StackExchange.

Now, #1 is the number of the current column and we have to draw a line from the column #2 to the column #3 (both included).

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

292 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
293 {
294     \tl_if_head_eq_meaning:nNTF { #2 } [
295         { #1 #2 }
296         { #1 { #2 } }
297     ]
298 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

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

302 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
303 {
304     \tl_set_rescan:Nno
305     #1
306     {
307         \char_set_catcode_other:N >

```

```

308     \char_set_catcode_other:N <
309   }
310   #1
311 }

```

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

```

312 \dim_new:N \l_@@_tmpc_dim
313 \dim_new:N \l_@@_tmpd_dim
314 \tl_new:N \l_@@_tmpc_tl
315 \tl_new:N \l_@@_tmpd_tl
316 \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.

```

317 \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).

```

318 \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.

```

319 \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.

```

320 \box_new:N \l_@@_the_array_box

```

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

```

321 \cs_new_protected:Npn \@@_qpoint:n #1
322 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

323 \bool_new:N \l_@@_tabular_bool

```

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

```

324 \bool_new:N \g_@@_delims_bool
325 \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.

```
326 \bool_new:N \l_@@_preamble_bool
327 \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.

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

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

```
329 \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.

```
330 \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).

```
331 \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.).

```
332 \dim_new:N \l_@@_col_width_dim
333 \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.

```
334 \int_new:N \g_@@_row_total_int
335 \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).

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

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

```
337 \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.

```
338 \tl_new:N \l_@@_hpos_cell_tl
339 \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`.

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

Idem for the mono-row blocks.

```
341 \dim_new:N \g_@@_blocks_ht_dim
342 \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}`).

```
343 \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`.

```
344 \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.

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

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

```
346 \bool_new:N \l_@@_notes_detect_duplicates_bool
347 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
348 \bool_new:N \l_@@_initial_open_bool
349 \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.

```
350 \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).

```
351 \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).

```
352 \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).

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

The value of the key `color-of-false`

```
354 \tl_new:N \l_@@_color_of_false_tl
355 \tl_set:Nn \l_@@_color_of_false_tl { nocolor }
```

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
```

The following flag will be raised when we are in a `\FalseRow`.

```
359 \bool_new:N \g_@@_FalseRow_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.

```
360 \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`).

```
361 \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`.

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

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

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

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

`\l_@@_bar_at_end_of_pream_bool` will be used for the construction of `\l_@@_array_preamble_tl`. It will be raised if you have a `|` at the end of the preamble provided by the final user and used during `@@_create_col_nodes:`.

```
365 \bool_new:N \l_@@_bar_at_end_of_pream_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 }`).

```
366 \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).

```
367 \bool_new:N \g_@@_aux_found_bool
```

In particular, in that `aux` file, there will be, for each environment of `nicematrix`, an affectation for the the following sequence that will contain information about the size of the array.

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

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

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

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

```
371 \tl_new:N \g_@@_user_preamble_tl
```

The token list `\l_@@_array_preamble_tl` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of `array`).

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

For `\multicolumn`.

```
373 \tl_new:N \g_@@_preamble_tl
```

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

```
374 \tl_new:N \l_@@_pre_cell_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`.

```
375 \tl_new:N \l_@@_columns_type_tl
376 \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 `:`.

```
377 \tl_new:N \l_@@_xdots_down_tl
378 \tl_new:N \l_@@_xdots_up_tl
379 \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`).

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

381 \cs_new_protected:Npn \@@_test_if_math_mode:
382 {
383   \if_mode_math: \else:
384     \@@_fatal:n { Outside-math-mode }
385   \fi:
386 }
```

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.

```
387 \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”.

```
388 \colorlet { nicematrix-last-col } { . }
389 \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`).

```
390 \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*.

```
391 \str_new:N \g_@@_com_or_env_str
392 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

```
393 \bool_new:N \l_@@_bold_row_style_bool
```

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

```
394 \clist_new:N \g_@@_cbic_clist
```

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

```
395 \clist_new:N \g_@@_col_with_trees_clist
```

The following parameter corresponds to the key `empty` of the command `\Block`.

```
396 \bool_new:N \l_@@_empty_block_bool
```

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

```

397 \cs_new:Npn \@@_full_name_env:
398 {
399   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
400   { command \space \c_backslash_str \g_@@_name_env_str }
401   { environment \space \{ \g_@@_name_env_str \} }
402 }

```

```

403 \tl_new:N \g_@@_cell_after_hook_tl

```

The argument is given by curryfication.

```

404 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
405 { \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.

```

406 \tl_new:N \g_@@_pre_code_before_tl
407 \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`.

```

408 \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.

```

409 \tl_new:N \g_@@_pre_code_after_tl
410 \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.

```

411 \bool_new:N \l_@@_in_code_after_bool

```

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

```

412 \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.

```

413 \int_new:N \l_@@_old_iRow_int
414 \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).

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

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

```
416 \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.

```
417 \bool_new:N \l_@@_X_columns_aux_bool
```

```
418 \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.

```
419 \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).

```
420 \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).

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

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

```
423 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
435 \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”.

```
436 \dim_new:N \g_@@_width_last_col_dim
437 \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.

```
438 \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!).

```
439 \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.

```
440 \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}`.

```
441 \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.

```
442 \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. In particular, it will be written on the file `aux`. However, for efficiency, the cells which are in a corner will also be marked directly in the main hash table of TeX.

```
443 \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`).

```
444 \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.

```
445 \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).

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

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

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

```
448 \int_new:N \l_@@_multiplicity_int
449 \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.

```
450 \int_new:N \g_@@_ddots_int
451 \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.

```
452 \dim_new:N \g_@@_delta_x_one_dim
453 \dim_new:N \g_@@_delta_y_one_dim
454 \dim_new:N \g_@@_delta_x_two_dim
455 \dim_new:N \g_@@_delta_y_two_dim
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code—before).

```
456 \int_new:N \l_@@_row_min_int
457 \int_new:N \l_@@_row_max_int
458 \int_new:N \l_@@_col_min_int
459 \int_new:N \l_@@_col_max_int

460 \int_new:N \l_@@_initial_i_int
461 \int_new:N \l_@@_initial_j_int
462 \int_new:N \l_@@_final_i_int
463 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
464 \int_new:N \l_@@_segment_start_int
465 \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.

```
466 \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).

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

The following parameters will store the “false columns” (with the letter F) in the preamble of the environment and the “false rows” created by `\FalseRow`.

```
468 \clist_new:N \g_@@_false_cols_clist
469 \clist_new:N \g_@@_false_rows_clist
```

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

```
470 \tl_new:N \l_@@_draw_tl
471 \seq_new:N \l_@@_tikz_seq
472 \tl_new:N \l_@@_tikz_rule_tl
```

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

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`).

```

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

477 \bool_new:N \l_@@_fix_vertex_bool

```

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

```

478 \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).

```

479 \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.

```

480 \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}`).

```

481 \bool_new:N \l_@@_hvlines_bool

```

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

```

482 \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`).

```

483 \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.

```

484 \int_new:N \l_@@_first_row_int
485 \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.

```

486 \int_new:N \l_@@_first_col_int
487 \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).

```
488 \int_new:N \l_@@_last_row_int
489 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
491 \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`.

```
492 \int_new:N \l_@@_last_col_int
493 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
494 \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`).

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

Some utilities

```
496 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
497 {
```

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.

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

```

498   \def \l_tmpa_tl { #1 }
499   \def \l_tmpb_tl { #2 }
500 }

```

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`.

```

501 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
502 {
503   \clist_if_in:NnF #1 { all }
504   {
505     \clist_clear:N \l_tmpa_clist
506     \clist_map_inline:Nn #1
507     {
508       \tl_if_head_eq_meaning:nNTF { ##1 } -
509       {

```

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.

```

510       \int_if_zero:nF { #2 }
511       {
512         \clist_put_right:Ne \l_tmpa_clist
513         { \int_eval:n { #2 + (##1) + 1 } }
514       }
515     }
516   {

```

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

```

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

```

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

```

520       \def \l_tmpa_tl { ##1 }
521       \def \l_tmpb_tl { ##1 }
522     }
523     \int_step_tokens:nnn \l_tmpa_tl \l_tmpb_tl
524     { \clist_put_right:Nn \l_tmpa_clist }
525   }
526 }
527 \tl_set_eq:NN #1 \l_tmpa_clist
528 }
529 }

```

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.

```

530 \AtBeginDocument
531 {
532   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
533   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
534 }

```

5 The command `\tabularnote`

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

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

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

```
535 \newcounter { tabularnote }
```

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

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

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

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

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

```
541 \seq_new:N \l_@@_notes_labels_seq
542 \newcounter { nicematrix_draft }
```

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

```

543 \cs_new_protected:Npn \@@_notes_format:n #1
544 {
545   \setcounter { nicematrix_draft } { #1 }
546   \@@_notes_style:n { nicematrix_draft }
547 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

551 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }

```

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

```

552 \NewDocumentCommand { \tabularnote } { o m }
553 { \@@_err_enumitem_not_loaded: }
554 \AddToHook { package / enumitem / after }
555 {

```

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

```

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

```

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

```

575   \RenewDocumentCommand { \tabularnote } { o m }
576   {
577     \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }

```

```

578     {
579         \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
580         { \@@_error:n { tabularnote~forbidden } }
581     }

```

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

```

582         \bool_if:NTF \l_@@_in_caption_bool
583         {
584             \tl_if_novalue:nTF { #1 }
585             { \@@_tabularnote_caption:nn { #1 } }
586             { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
587         }
588         {
589             \tl_if_novalue:nTF { #1 }
590             { \@@_tabularnote:nn { #1 } }
591             { \@@_tabularnote:nn { \exp_not:n { #1 } } }
592         }
593     { #2 }
594 }
595 }
596 }
597 }

598 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
599 {
600     \@@_error_or_warning:n { enumitem~not~loaded }
601     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
602 }

603 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
604 { \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`.

```

605 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
606 {

```

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).

```

607     \int_zero:N \l_tmpa_int
608     \bool_if:NT \l_@@_notes_detect_duplicates_bool
609     {

```

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`.

```

610     \int_zero:N \l_tmpb_int
611     \seq_map_indexed_inline:Nn \g_@@_notes_seq
612     {
613         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
614         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
615         {
616             \tl_if_novalue:nTF { #1 }
617             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
618             { \int_set:Nn \l_tmpa_int { ##1 } }

```

```

619         \seq_map_break:
620     }
621 }
622 \int_if_zero:nF \l_tmpa_int
623 { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
624 }
625 \int_if_zero:nT \l_tmpa_int
626 {
627     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
628     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
629 }
630 \seq_put_right:Ne \l_@@_notes_labels_seq
631 {
632     \tl_if_novalue:nTF { #1 }
633     {
634         \@@_notes_format:n
635         {
636             \int_eval:n
637             {
638                 \int_if_zero:nTF \l_tmpa_int
639                 \c@tabularnote
640                 \l_tmpa_int
641             }
642         }
643     }
644     { #1 }
645 }
646 \peek_meaning:NF \tabularnote
647 {

```

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`.

```

648     \hbox_set:Nn \l_tmpa_box
649     {

```

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

```

650         \@@_notes_label_in_tabular:n
651         { \seq_use:Nn \l_@@_notes_labels_seq { , } }
652     }

```

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

```

653     \int_gdecr:N \c@tabularnote
654     \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.

```

655     \int_gincr:N \g_@@_tabularnote_int
656     \refstepcounter { tabularnote }
657     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
658     { \int_gincr:N \c@tabularnote }
659     \seq_clear:N \l_@@_notes_labels_seq
660     \bool_lazy_or:nnTF
661     { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
662     { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
663     {
664         \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).

```

665         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
666     }

```

```

667         { \box_use:N \l_tmpa_box }
668     }
669 }

```

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`.

```

670 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
671 {
672     \bool_if:NTF \g_@@_caption_finished_bool
673     {
674         \int_compare:nNt \c@tabularnote = \g_@@_notes_caption_int
675         { \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`!

```

676         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
677         { \@@_error:n { Identical-notes-in-caption } }
678     }
679 }

```

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

```

680     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
681     {

```

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.

```

682         \bool_gset_true:N \g_@@_caption_finished_bool
683         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
684         \int_gzero:N \c@tabularnote
685     }
686     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
687 }

```

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!

```

688     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
689     \seq_put_right:Ne \l_@@_notes_labels_seq
690     {
691         \tl_if_novalue:nTF { #1 }
692         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
693         { #1 }
694     }
695     \peek_meaning:NF \tabularnote
696     {
697         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
698         \seq_clear:N \l_@@_notes_labels_seq
699     }
700 }

701 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
702 { \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.

```

703 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
704 {
705   \begin { pgfscope }
706   \pgfset
707   {
708     inner~sep = \c_zero_dim ,
709     minimum~size = \c_zero_dim
710   }
711   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
712   \pgfnode
713   { rectangle }
714   { center }
715   {
716     \vbox_to_ht:nn
717     { \dim_abs:n { #5 - #3 } }
718     {
719       \vfill
720       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
721     }
722   }
723   { #1 }
724   { }
725   \end { pgfscope }
726 }

```

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.

```

727 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
728 {
729   \begin { pgfscope }
730   \pgfset
731   {
732     inner~sep = \c_zero_dim ,
733     minimum~size = \c_zero_dim
734   }
735   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
736   \pgfpointdiff { #3 } { #2 }
737   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
738   \pgfnode
739   { rectangle }
740   { center }
741   {
742     \vbox_to_ht:nn
743     { \dim_abs:n \l_tmpb_dim }
744     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
745   }
746   { #1 }
747   { }
748   \end { pgfscope }
749 }

```

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).

```
750 \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.

```
751 \dim_new:N \l_@@_xdots_inter_dim
752 \AtBeginDocument
753 { \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).

```
754 \dim_new:N \l_@@_xdots_shorten_start_dim
755 \dim_new:N \l_@@_xdots_shorten_end_dim
756 \AtBeginDocument
757 {
758   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
759   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
760 }
```

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.

```
761 \dim_new:N \l_@@_xdots_radius_dim
762 \AtBeginDocument
763 { \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.

```
764 \tl_new:N \l_@@_xdots_line_style_tl
765 \tl_const:Nn \c_@@_standard_tl { standard }
766 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
767 \bool_new:N \l_@@_light_syntax_bool
768 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

```
769 \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).

```
770 \bool_new:N \l_@@_auto_columns_width_bool
771 \bool_new:N \l_@@_row_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!

```
772 \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.

```
773 \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”.

```
774 \bool_new:N \l_@@_medium_nodes_bool
775 \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).

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

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

```
777 \dim_new:N \l_@@_width_of_false_dim
778 \dim_set:Nn \l_@@_width_of_false_dim { 2 pt }
```

```
779 \keys_define:nn { nicematrix / xdots }
780 {
```

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.).

```
781     nullify .bool_set:N = \l_@@_nullify_dots_bool ,
782     brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
783     brace-shift+ .code:n =
784       \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
785     brace-shift+ .value_required:n = true ,
786     brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
787     Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
788     shorten-start .code:n =
789       \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
790     shorten-start .value_required:n = true ,
791     shorten-start+ .code:n =
792       \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
793     shorten-start~+ .meta:n = { shorten-start += #1 } ,
794     shorten-start+ .value_required:n = true ,
795     shorten-end .code:n =
796       \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
797     shorten-end .value_required:n = true ,
798     shorten-end+ .code:n =
799       \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
800     shorten-end+ .value_required:n = true ,
801     shorten-end~+ .meta:n = { shorten-end += #1 } ,
802     shorten .code:n =
803       \AtBeginDocument
804       {
805         \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
806         \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
807       } ,
808     shorten .value_required:n = true ,
809     shorten+ .code:n =
810       \AtBeginDocument
811       {
812         \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
813         \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
814       } ,
815     shorten~+ .meta:n = { shorten+ = #1 } ,
816     horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
817     horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
```

```

818   line-style .code:n =
819   {
820     \bool_lazy_or:nnTF
821     { \cs_if_exist_p:N \tikzpicture }
822     { \str_if_eq_p:nn { #1 } { standard } }
823     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
824     { \@@_error:n { bad-option-for-line-style } }
825   } ,
826   line-style .value_required:n = true ,
827   color .tl_set:N = \l_@@_xdots_color_tl ,
828   color .value_required:n = true ,
829   radius .code:n =
830     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
831   radius .value_required:n = true ,
832   inter .code:n =
833     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
834   radius .value_required:n = true ,

```

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

```

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

```

838   draw-first .code:n = \prg_do_nothing: ,
839   unknown .code:n =
840     \@@_unknown_key:nn { nicematrix / xdots } { Unknown-key-for-xdots }
841 }

```

```

842 \keys_define:nn { nicematrix / trees }
843 {
844   color.tl_set:N = \l_@@_trees_color_tl ,
845   color .value_required:n = true ,
846   line-width .dim_set:N = \l_@@_trees_line_width_dim ,
847   rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
848   rounded-corners .default:n = 2 pt ,
849   unknown .code:n =
850     \@@_unknown_key:nn { nicematrix / trees } { Unknown-key-for-trees }
851 }

```

```

852 \keys_define:nn { nicematrix / rules }
853 {
854   fix-vertex .code:n =
855     \bool_set_true:N \l_@@_fix_vertex_bool
856     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
857     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
858   color .tl_set:N = \l_@@_rules_color_tl ,
859   color .value_required:n = true ,
860   width .dim_set:N = \arrayrulewidth ,
861   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
862 }

```

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

```

863 \keys_define:nn { nicematrix / Global }
864 {
865   width-of-false .dim_set:N = \l_@@_width_of_false_dim ,

```

```

866 width-of-false .value_required:n = true ,
867 width-of-false+ .code:n =
868   \dim_add:Nn \l_@@_width_of_false_dim { #1 } ,
869 width-of-false+ .value_required:n = true ,
870 width-of-false~+ .meta:n = { width-of-false+ = #1 } ,
871 color-of-false .tl_set:N = \l_@@_color_of_false_tl ,
872 color-of-false .value_required:n = true ,
873 fix-vertex .value_forbidden:n = true ,
874 brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
875 brace-shift+ .code:n =
876   \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
877 brace-shift+ .value_required:n = true ,
878 brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
879 caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
880 show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
881 color-inside .code:n = \@@_fatal:n { key~color-inside } ,
882 colortbl-like .code:n = \@@_fatal:n { key~color-inside } ,
883 ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
884 &-in-blocks .meta:n = ampersand-in-blocks ,
885 no-cell-nodes .choices:nn = { true , false }
886 {
887   \tl_if_eq:NnTF \l_keys_choice_tl { true }
888   {
889     \bool_set_true:N \l_@@_no_cell_nodes_bool
890     \cs_set_eq:NN \@@_node_cell: \@@_node_cell_inactive:
891   }
892   {
893     \bool_set_false:N \l_@@_no_cell_nodes_bool
894     \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
895   }
896 } ,
897 no-cell-nodes .default:n = true ,

```

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

```

898 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
899 rounded-corners .default:n = 4 pt ,
900 custom-line .code:n = \@@_custom_line:n { #1 } ,
901 default-line .code:n = \@@_default_line:n { #1 } ,
902 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
903 rules .value_required:n = true ,
904 trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
905 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`.

```

906 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
907 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
908 cell-space-top-limit+ .code:n =
909   \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
910 cell-space-top-limit+ .value_required:n = true ,
911 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
912 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
913 cell-space-bottom-limit+ .code:n =
914   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
915 cell-space-bottom-limit+ .value_required:n = true ,
916 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
917 cell-space-limits .meta:n =
918 {
919   cell-space-top-limit = #1 ,
920   cell-space-bottom-limit = #1 ,
921 } ,
922 cell-space-limits .value_required:n = true ,

```

```

923 cell-space-limits+ .meta:n =
924 {
925     cell-space-top-limit += #1 ,
926     cell-space-bottom-limit += #1 ,
927 } ,
928 cell-space-limits+ .value_required:n = true ,
929 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
930 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
931 light-syntax .choices:nn = { true , false }
932 {
933     \tl_if_eq:VnTF \l_keys_value_tl { true }
934     { \bool_set_true:N \l_@@_light_syntax_bool }
935     { \bool_set_false:N \l_@@_light_syntax_bool }
936     \bool_set_false:N \l_@@_light_syntax_expanded_bool
937 } ,
938 light-syntax .default:n = true ,
939 light-syntax-expanded .choices:nn = { true , false }
940 {
941     \tl_if_eq:VnTF \l_keys_value_tl { true }
942     {
943         \bool_set_true:N \l_@@_light_syntax_bool
944         \bool_set_true:N \l_@@_light_syntax_expanded_bool
945     }
946     {
947         \bool_set_false:N \l_@@_light_syntax_bool
948         \bool_set_false:N \l_@@_light_syntax_expanded_bool
949     }
950 } ,
951 light-syntax-expanded .default:n = true ,

```

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

```

952 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
953 end-of-row .value_required:n = true ,
954 end-of-row .initial:n = ; ,
955
956 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
957 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
958 last-row .int_set:N = \l_@@_last_row_int ,
959 last-row .default:n = -1 ,
960
961 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
962 code-for-first-col .value_required:n = true ,
963 code-for-first-col+ .code:n =
964 { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
965 code-for-first-col+ .value_required:n = true ,
966 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
967
968 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
969 code-for-last-col .value_required:n = true ,
970 code-for-last-col+ .code:n =
971 { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
972 code-for-last-col+ .value_required:n = true ,
973 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
974
975 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
976 code-for-first-row .value_required:n = true ,
977 code-for-first-row+ .code:n =
978 { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
979 code-for-first-row+ .value_required:n = true ,
980 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
981
982 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
983 code-for-last-row .value_required:n = true ,

```

```

984 code-for-last-row+ .code:n =
985   { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
986 code-for-last-row+ .value_required:n = true ,
987 code-for-last-row+ .meta:n = { code-for-last-row+ = #1 } ,
988
989 hlines .clist_set:N = \l_@@_hlines_clist ,
990 hlines .default:n = all ,
991 vlines .clist_set:N = \l_@@_vlines_clist ,
992 vlines .default:n = all ,
993
994 vl_lines-in-sub-matrix .code:n =
995   {
996     \tl_if_single_token:nTF { #1 }
997     {
998       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
999       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

1000   { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
1001   }
1002   { \@@_error:n { One-letter-allowed } }
1003   } ,
1004 vl_lines-in-sub-matrix .value_required:n = true ,
1005 hv_lines .code:n =
1006   {
1007     \bool_set_true:N \l_@@_hv_lines_bool
1008     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1009     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1010   } ,
1011 hv_lines .value_forbidden:n = true ,
1012 hv_lines-except-borders .code:n =
1013   {
1014     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
1015     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1016     \bool_set_true:N \l_@@_hv_lines_bool
1017     \bool_set_true:N \l_@@_except_borders_bool
1018   } ,
1019 hv_lines-except-borders .value_forbidden:n = true ,
1020 hv_lines-except-borders .code:n =
1021   {
1022     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
1023     \bool_set_true:N \l_@@_hlines_bool
1024     \bool_set_true:N \l_@@_except_borders_bool
1025   } ,
1026 hv_lines-except-borders .value_forbidden:n = true ,
1027 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1028 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.

```

1029 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1030 renew-dots .value_forbidden:n = true ,
1031 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1032 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1033 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1034 create-extra-nodes .meta:n =
1035   { create-medium-nodes , create-large-nodes } ,
1036 left-margin .dim_set:N = \l_@@_left_margin_dim ,
1037 left-margin .default:n = \arraycolsep ,
1038 right-margin .dim_set:N = \l_@@_right_margin_dim ,
1039 right-margin .default:n = \arraycolsep ,
1040 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1041 margin .default:n = \arraycolsep ,

```

```

1042     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1043     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1044     extra-margin .meta:n =
1045       { extra-left-margin = #1 , extra-right-margin = #1 } ,
1046     extra-margin .value_required:n = true ,
1047     respect-arraystretch .code:n =
1048       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1049     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.

```

1050     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1051     pgf-node-code .value_required:n = true ,
1052   }

```

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

```

1053 \keys_define:nn { nicematrix / environments }
1054 {
1055   draw-trees-in-col .code:n =
1056     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1057   draw-trees-in-col .value_required:n = true ,
1058   create-blocks-in-col .code:n =
1059     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1060   create-blocks-in-col .value_required:n = true ,
1061   corners .clist_set:N = \l_@@_corners_clist ,
1062   corners .default:n = { NW , SW , NE , SE } ,
1063   code-before .code:n =
1064     {
1065       \tl_if_empty:nF { #1 }
1066       {
1067         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1068         \bool_set_true:N \l_@@_code_before_bool
1069       }
1070     } ,
1071   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}`.

```

1072   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1073   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1074   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).

```

1075   baseline .tl_set:N = \l_@@_baseline_tl ,
1076   baseline .value_required:n = true ,
1077   baseline .initial:n = c ,
1078   columns-width .code:n =
1079     \str_case:nnF { #1 }
1080     {
1081       { auto } { \bool_set_true:N \l_@@_auto_columns_width_bool }
1082       { row-height } { \bool_set_true:N \l_@@_row_columns_width_bool }
1083     }
1084     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1085   columns-width .value_required:n = true ,
1086   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.

```

1087   \legacy_if:nF { measuring@ }
1088   {

```

```

1089     \str_set:Ne \l_@@_name_str { #1 }
1090     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1091     { \@@_err_duplicate_names:n { #1 } }
1092     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1093   } ,
1094   name .value_required:n = true ,
1095   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1096   code-after .value_required:n = true ,
1097 }
1098 \cs_set:Npn \@@_err_duplicate_names:n #1
1099 { \@@_error:nn { Duplicate-name } { #1 } }
1100 \keys_define:nn { nicematrix / notes }
1101 {
1102   no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1103   para .bool_set:N = \l_@@_notes_para_bool ,
1104   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1105   code-before .value_required:n = true ,
1106   code-before+ .code:n =
1107     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1108   code-before+ .value_required:n = true ,
1109   code-before~+ .code:n =
1110     \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1111   code-before~+ .value_required:n = true ,
1112   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1113   code-after .value_required:n = true ,
1114   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1115   style .cs_set:Np = \@@_notes_style:n #1 ,
1116   style .value_required:n = true ,
1117   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1118   label-in-tabular .value_required:n = true ,
1119   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1120   label-in-list .value_required:n = true ,
1121   enumitem-keys .code:n =
1122   {
1123     \AtBeginDocument
1124     {
1125       \IfPackageLoadedT { enumitem }
1126       { \setlist* [ tabularnotes ] { #1 } }
1127     }
1128   } ,
1129   enumitem-keys .value_required:n = true ,
1130   enumitem-keys-para .code:n =
1131   {
1132     \AtBeginDocument
1133     {
1134       \IfPackageLoadedT { enumitem }
1135       { \setlist* [ tabularnotes* ] { #1 } }
1136     }
1137   } ,
1138   enumitem-keys-para .value_required:n = true ,
1139   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1140   unknown .code:n =
1141     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1142 }

```

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.

```

1143 \keys_define:nn { nicematrix / delimiters }
1144 {
1145   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1146   color .tl_set:N = \l_@@_delimiters_color_tl ,
1147   color .value_required:n = true ,
1148 }

```

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

```

1149 \keys_define:nn { nicematrix }
1150 {
1151   NiceMatrixOptions .inherit:n =
1152     { nicematrix / Global } ,
1153   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1154   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1155   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1156   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1157   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1158   SubMatrix / rules .inherit:n = nicematrix / rules ,
1159   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1160   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1161   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1162   NiceMatrix .inherit:n =
1163     {
1164       nicematrix / Global ,
1165       nicematrix / environments ,
1166     } ,
1167   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1168   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1169   NiceMatrix / trees .inherit:n = nicematrix / trees ,
1170   NiceTabular .inherit:n =
1171     {
1172       nicematrix / Global ,
1173       nicematrix / environments
1174     } ,
1175   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1176   NiceTabular / rules .inherit:n = nicematrix / rules ,
1177   NiceTabular / notes .inherit:n = nicematrix / notes ,
1178   NiceTabular / trees .inherit:n = nicematrix / trees ,
1179   NiceArray .inherit:n =
1180     {
1181       nicematrix / Global ,
1182       nicematrix / environments ,
1183     } ,
1184   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1185   NiceArray / rules .inherit:n = nicematrix / rules ,
1186   NiceArray / trees .inherit:n = nicematrix / trees ,
1187   pNiceArray .inherit:n =
1188     {
1189       nicematrix / Global ,
1190       nicematrix / environments ,
1191     } ,
1192   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1193   pNiceArray / rules .inherit:n = nicematrix / rules ,
1194   pNiceArray / trees .inherit:n = nicematrix / trees
1195 }

```

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

```

1196 \keys_define:nn { nicematrix / NiceMatrixOptions }
1197 {
1198   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

1199   delimiters / color .value_required:n = true ,
1200   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1201   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1202   delimiters .value_required:n = true ,
1203   width .dim_set:N = \l_@@_width_dim ,
1204   last-col .code:n =
1205     \tl_if_empty:nF { #1 }
1206       { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1207       \int_zero:N \l_@@_last_col_int ,
1208   small .bool_set:N = \l_@@_small_bool ,
1209   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.

```

1210   renew-matrix .code:n = \@@_renew_matrix: ,
1211   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}`.

```

1212   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.

```

1213   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)`.

```

1214     \str_if_eq:eeTF { #1 } { auto }
1215     { \@@_error:n { Option-auto-for-columns-width } }
1216     { \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.

```

1217   allow-duplicate-names .code:n =
1218     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1219   allow-duplicate-names .value_forbidden:n = true ,
1220   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1221   notes .value_required:n = true ,
1222   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1223   sub-matrix .value_required:n = true ,
1224   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1225   matrix / columns-type .value_required:n = true ,
1226   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1227   unknown .code:n =
1228     \@@_unknown_key:nn
1229     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1230     { Unknown-key-for-NiceMatrixOptions }
1231 }

```

`\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.

```

1232 \NewDocumentCommand \NiceMatrixOptions { m }
1233 { \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.

```

1234 \keys_define:nn { nicematrix / NiceMatrix }
1235 {
1236   last-col .code:n = \tl_if_empty:nTF { #1 }

```

```

1237         {
1238             \bool_set_true:N \l_@@_last_col_without_value_bool
1239             \int_set:Nn \l_@@_last_col_int { -1 }
1240         }
1241         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1242     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1243     columns-type .value_required:n = true ,
1244     l .meta:n = { columns-type = l } ,
1245     r .meta:n = { columns-type = r } ,
1246     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1247     delimiters / color .value_required:n = true ,
1248     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1249     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1250     delimiters .value_required:n = true ,
1251     small .bool_set:N = \l_@@_small_bool ,
1252     small .value_forbidden:n = true ,
1253     unknown .code:n =
1254         \@@_unknown_key:nn
1255             { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1256             { Unknown-key-for~NiceMatrix }
1257 }

```

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

```

1258 \keys_define:nn { nicematrix / NiceArray }
1259 {

```

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.

```

1260     small .bool_set:N = \l_@@_small_bool ,
1261     small .value_forbidden:n = true ,
1262     last-col .code:n = \tl_if_empty:nF { #1 }
1263         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1264         \int_zero:N \l_@@_last_col_int ,
1265     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1266     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1267     unknown .code:n =
1268         \@@_unknown_key:nn
1269             { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1270             { Unknown-key-for~NiceArray }
1271 }

1272 \keys_define:nn { nicematrix / pNiceArray }
1273 {
1274     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1275     last-col .code:n = \tl_if_empty:nF { #1 }
1276         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1277         \int_zero:N \l_@@_last_col_int ,
1278     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1279     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1280     delimiters / color .value_required:n = true ,
1281     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1282     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1283     delimiters .value_required:n = true ,
1284     small .bool_set:N = \l_@@_small_bool ,
1285     small .value_forbidden:n = true ,
1286     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1287     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1288     unknown .code:n =
1289         \@@_unknown_key:nn
1290             { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1291             { Unknown-key-for~NiceMatrix }
1292 }

```

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

```
1293 \keys_define:nn { nicematrix / NiceTabular }
1294 {
```

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.

```
1295   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1296           \bool_set_true:N \l_@@_width_used_bool ,
1297   width .value_required:n = true ,
1298   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1299   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1300   tabularnote .value_required:n = true ,
1301   caption .tl_set:N = \l_@@_caption_tl ,
1302   caption .value_required:n = true ,
1303   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1304   short-caption .value_required:n = true ,
1305   label .tl_set:N = \l_@@_label_tl ,
1306   label .value_required:n = true ,
1307   last-col .code:n = \tl_if_empty:nF { #1 }
1308           { \@@_error:n { last-col-non-empty-for-NiceArray } }
1309           \int_zero:N \l_@@_last_col_int ,
1310   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1311   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1312   unknown .code:n =
1313     \@@_unknown_key:nn
1314     { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1315     { Unknown-key-for-NiceTabular }
1316 }
```

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
```

```
1317 \keys_define:nn { nicematrix / CodeAfter }
1318 {
1319   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1320   delimiters / color .value_required:n = true ,
1321   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1322   rules .value_required:n = true ,
1323   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1324   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1325   sub-matrix .value_required:n = true ,
1326   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1327 }
```

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}`).

```
1328 \cs_new_protected:Npn \@@_cell_begin:
1329 {
```

`\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.

```
1330 \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

```
1331 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1332 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

```
1333 \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:
```

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

```
1334 \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.

```
1335 \if_int_compare:w \c@jCol = \c_one_int
1336 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1337 \@@_begin_of_row:
1338 \fi:
1339 \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:`.

```
1340 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1341 \@@_tuning_not_tabular_begin:
1342 \@@_tuning_exterior_rows:
1343 \g_@@_row_style_tl
1344 }
1345 \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.

```
1346 \cs_new_protected:Npn \@@_tuning_first_last_row:
1347 {
1348   \if_int_compare:w \c@iRow = \c_zero_int
1349   \if_int_compare:w \c@jCol > \c_zero_int
1350   \l_@@_code_for_first_row_tl
1351   \@@_tuning_key_small:
1352   \xglobal \colorlet { nicematrix-first-row } { . }
```

```

1353     \fi:
1354   \else:
1355     \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1356   \fi:
1357 }

```

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.

```

1358 \cs_new_protected:Npn \@@_tuning_last_row:
1359 {
1360   \if_int_compare:w \c@iRow = \l_1\@@_last_row_int
1361     \l_1\@@_code_for_last_row_tl
1362     \@@_tuning_key_small:
1363     \xglobal \colorlet { nicematrix-last-row } { . }
1364   \fi:
1365 }

```

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

```

1366 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1367 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1368 {
1369   \m@th
1370   $ \% $

```

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

```

1371   \@@_tuning_key_small:
1372 }
1373 \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.

```

1374 \cs_new_protected:Npn \@@_begin_of_row:
1375 {
1376   \int_gincr:N \c@iRow
1377   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1378   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1379   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1380   \pgfpicture
1381   \pgfrememberpicturerepositiononpagetrue
1382   \pgfcoordinate
1383     { \@@_env: - row - \int_use:N \c@iRow - base }
1384     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1385   \str_if_empty:NF \l_1\@@_name_str
1386   {
1387     \pgfnodealias
1388       { \l_1\@@_name_str - row - \int_use:N \c@iRow - base }
1389       { \@@_env: - row - \int_use:N \c@iRow - base }
1390   }
1391   \endpgfpicture
1392 }

```

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.

```
1393 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1394 {
1395   \if_int_compare:w \c@iRow = \c_zero_int
1396     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1397       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1398     \fi:
1399     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1400       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1401     \fi:
1402   \else:
1403     \if_int_compare:w \c@iRow = \c_one_int
1404       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1405         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1406       \fi:
1407     \fi:
1408   \fi:
1409 }

1410 \cs_new_protected:Npn \@@_rotate_cell_box:
1411 {
1412   \box_rotate:Nn \l_@@_cell_box
1413     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1414   \bool_if:NTF \g_@@_rotate_c_bool
1415     {
1416       \hbox_set:Nn \l_@@_cell_box
1417         { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1418     }
1419   {
1420     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1421     {
1422       \vbox_set_top:Nn \l_@@_cell_box
1423       {
```

```

1424         \vbox_to_zero:n { }
1425         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1426         \box_use:N \l_@@_cell_box
1427     }
1428 }
1429 }
1430 \bool_gset_false:N \g_@@_rotate_bool
1431 \bool_gset_false:N \g_@@_rotate_c_bool
1432 \bool_gset_false:N \g_@@_rotate_minus_bool
1433 }

```

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

```

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

```

Here is a version slightly more efficient.

```

1434 \cs_set_protected:Npn \@@_adjust_size_box:
1435 {
1436   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1437   \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1438     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1439   \fi:
1440   \global \g_@@_blocks_wd_dim = \c_zero_dim
1441 \fi:
1442   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1443   \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1444     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1445   \fi:
1446   \global \g_@@_blocks_dp_dim = \c_zero_dim
1447 \fi:
1448   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1449   \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1450     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1451   \fi:
1452   \global \g_@@_blocks_ht_dim = \c_zero_dim
1453 \fi:
1454 }
1455 \cs_new_protected:Npn \@@_cell_end:
1456 {

```

The following command is nullified in the tabulars.

```

1457 \@@_tuning_not_tabular_end:
1458 \hbox_set_end:

```

```

1459     \@@_cell_end_i:
1460 }

\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.

```

1461 \cs_new_protected:Npn \@@_cell_end_i:
1462 {

```

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.

```

1463   \g_@@_cell_after_hook_tl
1464   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1465   \@@_adjust_size_box:
1466   \bool_if:NF \g_@@_FalseRow_bool
1467   {
1468     \box_set_ht:Nn \l_@@_cell_box
1469       { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1470     \box_set_dp:Nn \l_@@_cell_box
1471       { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
1472   }

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the potential “first column” and the potential “last column”).

```

1473   \@@_update_max_cell_width:

```

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

```

1474   \@@_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.

```

1475 \bool_if:NTF \g_@@_empty_cell_bool
1476 { \box_use_drop:N \l_@@_cell_box }
1477 {
1478   \bool_if:NTF \g_@@_not_empty_cell_bool
1479   \@@_print_node_cell:
1480   {
1481     \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1482     \@@_print_node_cell:
1483     \else:
1484       \box_use_drop:N \l_@@_cell_box
1485     \fi:
1486   }
1487 }
1488 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1489 \global \g_@@_col_total_int = \c@jCol
1490 \fi:
1491 \global \let \g_@@_empty_cell_bool \c_false_bool
1492 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1493 }

```

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:

```

1494 \cs_new_protected:Npn \@@_update_max_cell_width:
1495 {
1496   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1497   \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1498   \fi:
1499 }

```

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`).

```

1500 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1501 {
1502   \@@_math_toggle:
1503   \hbox_set_end:
1504   \bool_if:NF \g_@@_rotate_bool
1505   {
1506     \hbox_set:Nn \l_@@_cell_box
1507     {
1508       \makebox [ \l_@@_col_width_dim ] [ s ]
1509       { \hbox_unpack_drop:N \l_@@_cell_box }
1510     }
1511   }

```

```

1511     }
1512     \@@_cell_end_i:
1513 }

1514 \pgfset
1515 {
1516     nicematrix / cell-node /.style =
1517     {
1518         inner~sep = \c_zero_dim ,
1519         minimum~width = \c_zero_dim
1520     }
1521 }

```

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.

```

1522 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1523 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1524 {
1525     \use:c
1526     {
1527         __siunitx_table_align_
1528         \bool_if:NTF \l__siunitx_table_text_bool
1529         \l__siunitx_table_align_text_tl
1530         \l__siunitx_table_align_number_str
1531         :n
1532     }
1533     { #1 }
1534 }

```

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!

```

1535 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1536 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1537 {
1538     \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1539     \hbox:n
1540     {
1541         \pgfsys@markposition
1542         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1543     }
1544     #1
1545     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1546     \hbox:n
1547     {
1548         \pgfsys@markposition
1549         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1550     }
1551 }

```

```

1552 \cs_new_protected:Npn \@@_print_node_cell:
1553 {
1554     \socket_use:nn { nicematrix / siunitx-wrap }
1555     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1556 }

```

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

```

1557 \cs_new_protected:Npn \@@_node_cell_active:
1558 {
1559   \pgfpicture
1560   \pgfsetbaseline \c_zero_dim
1561   \pgfrememberpicturepositiononpagetrue
1562   \pgfset { nicematrix / cell-node }
1563   \pgfnode
1564     { rectangle }
1565     { base }
1566     {

```

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).

```

1567     \sys_if_engine_xetex:T { \set@color }
1568     \box_use:N \l_@@_cell_box
1569   }
1570   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1571   { \l_@@_pgf_node_code_tl }
1572   \str_if_empty:NF \l_@@_name_str
1573   {
1574     \pgfnodealias
1575       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1576       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1577   }
1578   \endpgfpicture
1579 }
1580 \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1581 \cs_new_protected:Npn \@@_node_cell_inactive:
1582 { \set@color \box_use_drop:N \l_@@_cell_box }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1583 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1584 {
1585   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1586     { g_@@_ #2 _ lines _ tl }
1587   {
1588     \use:c { @@ _ draw _ #2 : nnn }
1589     { \int_use:N \c@iRow }
1590     { \int_use:N \c@jCol }
1591     { \exp_not:n { #3 } }
1592   }
1593 }

```

```

1594 \cs_new_protected:Npn \l_@@_array:n
1595 {
1596   \dim_set:Nn \col@sep
1597   {
1598     \bool_if:NTF \l_@@_row_columns_width_bool
1599     { \c_zero_dim }
1600     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1601   }
1602   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1603   { \def \@halignto { } }
1604   { \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.

```

1605   \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
1606   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1607 }
1608 \cs_generate_variant:Nn \l_@@_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.

```

1609 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign

```

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

```

1610 \cs_new_protected:Npn \l_@@_create_row_node:
1611 {
1612   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1613   {
1614     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1615     \l_@@_create_row_node_i:
1616   }
1617 }
1618 \cs_new_protected:Npn \l_@@_create_row_node_i:
1619 {

```

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

```

1620   \hbox
1621   {
1622     \bool_if:NT \l_@@_code_before_bool
1623     {
1624       \vtop
1625       {
1626         \skip_vertical:N 0.5\arrayrulewidth
1627         \pgfsys@markposition
1628         { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1629         \skip_vertical:N -0.5\arrayrulewidth
1630       }
1631     }
1632     \pgfpicture
1633     \pgfrememberpicturerepositiononpagetrue
1634     \pgfcoordinate { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1635     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1636     \str_if_empty:NF \l_@@_name_str
1637     {
1638       \pgfnodealias
1639       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1640       { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1641     }

```

```

1642     \endpgfpicture
1643   }
1644 }

1645 \cs_new_protected:Npn \@@_in_everycr:
1646 {
1647   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1648   \tbl_update_cell_data_for_next_row:
1649   \int_gzero:N \c@jCol
1650   \bool_gset_false:N \g_@@_after_col_zero_bool
1651   \bool_if:NF \g_@@_row_of_col_done_bool
1652   {
1653     \@@_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).

```

1654     \clist_if_empty:NF \l_@@_hlines_clist
1655     {
1656       \str_if_eq:eeF \l_@@_hlines_clist { all }
1657       {
1658         \clist_if_in:NeT
1659         \l_@@_hlines_clist
1660         { \int_eval:n { \c@iRow + 1 } }
1661       }
1662     }

```

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.

```

1663       \int_compare:nNnT \c@iRow > { -1 }
1664       {
1665         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1666         { \hrule height \arrayrulewidth width \c_zero_dim }
1667       }
1668     }
1669   }
1670 }
1671 }

```

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

```

1672 \cs_set_protected:Npn \@@_renew_dots:
1673 {
1674   \cs_set_eq:NN \ldots \@@_Ldots:
1675   \cs_set_eq:NN \cdots \@@_Cdots:
1676   \cs_set_eq:NN \vdots \@@_Vdots:
1677   \cs_set_eq:NN \ddots \@@_Ddots:
1678   \cs_set_eq:NN \iddots \@@_Iddots:
1679   \cs_set_eq:NN \dots \@@_Ldots:
1680   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1681 }

```

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.

```

1682 \cs_new_protected:Npn \@@_patch_booktabs: { }
1683 \AddToHook { package / booktabs / after }

```

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

```

1684 {
1685   \cs_set_protected:Npn \@@_patch_booktabs:
1686     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1687 }

```

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`.

```

1688 \cs_new_protected:Npn \@@_some_initialization:
1689 {
1690   \@@_everycr:
1691   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1692   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1693   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1694   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1695   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1696   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1697 }

```

`\@@_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`.

```

1698 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1699 {

```

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

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

```

1700   \seq_gclear:N \g_@@_pos_of_blocks_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.

```

1701   \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.

```

1702   \fp_gzero:N \g_@@_total_X_weight_fp
1703   \bool_gset_false:N \g_@@_V_of_X_bool
1704   \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1705   \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1706   \@@_patch_booktabs:
1707   \box_clear_new:N \l_@@_cell_box
1708   \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}`).

```

1709   \bool_if:NT \l_@@_small_bool
1710   {

```

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

```

1711     \def \arraystretch { 0.47 }
1712     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1713     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1714 }

```

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!

```

1715     \bool_if:NT \g_@@_create_cell_nodes_bool
1716     {
1717         \tbl_put_right:Nn \@@_begin_of_row:
1718         {
1719             \pgfsys@markposition
1720             { \@@_env: - row - \int_use:N \c@iRow - base }
1721         }
1722         \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1723     }

```

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`.

```

1724     \def \ar@ialign
1725     {
1726         \tbl_init_cell_data_for_table:
1727         \@@_some_initialization:
1728         \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`.

```

1729         \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1730         \halign
1731     }

```

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).

```

1732     \cs_set_eq:NN \@@_old_ldots: \ldots
1733     \cs_set_eq:NN \@@_old_cdots: \cdots
1734     \cs_set_eq:NN \@@_old_vdots: \vdots
1735     \cs_set_eq:NN \@@_old_ddots: \ddots
1736     \cs_set_eq:NN \@@_old_iddots: \iddots
1737     \bool_if:NTF \l_@@_standard_cline_bool
1738     { \cs_set_eq:NN \cline \@@_standard_cline: }
1739     { \cs_set_eq:NN \cline \@@_cline: }
1740     \bool_if:NTF \l_@@_no_cell_nodes_bool
1741     { \@@_redefine_xdots_no_cell_nodes: }
1742     { \@@_redefine_xdots: }
1743     \cs_set_eq:NN \Hline \@@_Hline:
1744     \cs_set_eq:NN \Hspace \@@_Hspace:
1745     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1746     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1747     \cs_set_eq:NN \Block \@@_Block:
1748     \cs_set_eq:NN \rotate \@@_rotate:
1749     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1750     \cs_set_eq:NN \dotfill \@@_dotfill:
1751     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1752     \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1753     \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }

```

```

1754 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1755 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1756 \cs_set_eq:NN \TopRule \@@_TopRule
1757 \cs_set_eq:NN \MidRule \@@_MidRule
1758 \cs_set_eq:NN \BottomRule \@@_BottomRule
1759 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1760 \cs_set_eq:NN \Hbrace \@@_Hbrace
1761 \cs_set_eq:NN \Vbrace \@@_Vbrace
1762 \cs_set_eq:NN \hdashedline \@@_hdashedline:
1763 \cs_set_eq:NN \cdashedline \@@_cdashedline:n
1764 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1765 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1766 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1767 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1768 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1769 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1770 \cs_set_eq:NN \FalseRow \@@_FalseRow
1771 \int_if_zero:nTF \l_@@_first_row_int
1772 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1773 {
1774   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1775   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1776 }
1777 \bool_if:NT \l_@@_renew_dots_bool { \@@_renew_dots: }

```

We redefine `\multicolumn`⁷.

```

1778 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the `aux` file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1779 \tl_if_exist:NT \l_@@_note_in_caption_tl
1780 {
1781   \tl_if_empty:NF \l_@@_note_in_caption_tl
1782   {
1783     \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1784     \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1785   }
1786 }

```

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).

```

1787 \seq_gclear:N \g_@@_multicolumn_cells_seq
1788 \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`).

```

1789 \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).

```

1790 \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`).

```

1791 \int_gzero:N \g_@@_row_total_int

```

⁷Since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we have used the hook `tabular/begin` to revert the definition.

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.

```

1792     \int_gzero:N \g_@@_col_total_int
1793     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1794     \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.

```

1795     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1796     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1797     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1798     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1799     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1800     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1801     \tl_gclear:N \g_nicematrix_code_before_tl
1802     \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]`).

```

1803     \dim_zero_new:N \l_@@_left_delim_dim
1804     \dim_zero_new:N \l_@@_right_delim_dim
1805     \bool_if:NTF \g_@@_delims_bool
1806     {

```

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

```

1807         \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1808         \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1809         \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1810         \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1811     }
1812     {
1813         \dim_gset:Nn \l_@@_left_delim_dim
1814         { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1815         \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1816     }
1817 }

```

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.

```

1818 \cs_new_protected:Npn \@@_pre_array:
1819 {
1820     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1821     \int_gzero_new:N \c@iRow
1822     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1823     \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).

```

1824     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1825     { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1826     \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1827     { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1828     \bool_if:NT \g_@@_aux_found_bool
1829     {
1830         \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }

```

```

1831 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1832 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1833 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1834 }

```

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).

```

1835 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1836 {
1837   \bool_set_true:N \l_@@_last_row_without_value_bool
1838   \bool_if:NT \g_@@_aux_found_bool
1839     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1840 }
1841 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1842 {
1843   \bool_if:NT \g_@@_aux_found_bool
1844     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1845 }

```

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”.

```

1846 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1847 {
1848   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1849     {
1850       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1851       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1852       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1853       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1854     }
1855 }

1856 \seq_gclear:N \g_@@_cols_vlism_seq
1857 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1858 \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1859 \@@_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).

```

1860 \hbox_set:Nw \l_@@_the_array_box
1861 \skip_horizontal:N \l_@@_left_margin_dim
1862 \skip_horizontal:N \l_@@_extra_left_margin_dim
1863 \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).

```

1864 \m@th
1865 $ % $
1866 \bool_if:NTF \l_@@_light_syntax_bool
1867   { \use:c { @@-light-syntax } }
1868   { \use:c { @@-normal-syntax } }
1869 }

```

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

```

1870 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1871 {
1872   \tl_set:Nn \l_tmpa_tl { #1 }
1873   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1874     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1875   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1876   \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...

```

1877   \@@_pre_array:
1878 }

```

```

1879 \cs_new_protected:Npn \@@_redefine_xdots:
1880 {
1881   \cs_set_eq:NN \Ldots \@@_Ldots:
1882   \cs_set_eq:NN \Cdots \@@_Cdots:
1883   \cs_set_eq:NN \Vdots \@@_Vdots:
1884   \cs_set_eq:NN \Ddots \@@_Ddots:
1885   \cs_set_eq:NN \Iddots \@@_Iddots:
1886 }
1887 \cs_new_protected:Npn \@@_redefine_xdots_no_cell_nodes:
1888 {
1889   \cs_set_protected:Npn \Ldots { \@@_error_xdots:N \Ldots }
1890   \cs_set_protected:Npn \Cdots { \@@_error_xdots:N \Cdots }
1891   \cs_set_protected:Npn \Vdots { \@@_error_xdots:N \Vdots }
1892   \cs_set_protected:Npn \Ddots { \@@_error_xdots:N \Ddots }
1893   \cs_set_protected:Npn \Iddots { \@@_error_xdots:N \Iddots }
1894 }
1895 \cs_new_protected:Npn \@@_error_xdots:N #1
1896 { \@@_error:nn { xdots~without~cell~nodes } { #1 } }

```

9 The `\CodeBefore`

```

1897 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1898 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Misuse~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).

```

1899 \cs_new_protected:Npn \@@_pre_code_before:
1900 {

```

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.

```

1901   \pgfsys@markposition { \@@_env: - position }
1902   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1903   \pgfpicture
1904   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1905   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1906   {
1907     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1908     \pgfcoordinate { \@@_env: - row - ##1 }
1909     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1910   }

```

Now, the recreation of the col nodes.

```

1911 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1912 {
1913   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1914   \pgfcoordinate { \@@_env: - col - ##1 }
1915   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1916 }

```

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

```

1917 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1918 \endpgfpicture

```

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

```

1919 \@@_create_diag_nodes:

```

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

```

1920 \@@_create_blocks_nodes:
1921 \IfPackageLoadedT { tikz }
1922 {
1923   \tikzset
1924   {
1925     every~picture / .style =
1926     { overlay , name~prefix = \@@_env: - }
1927   }
1928 }
1929 \cs_set_eq:NN \cellcolor \@@_cellcolor
1930 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1931 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1932 \cs_set_eq:NN \rowcolor \@@_rowcolor
1933 \cs_set_eq:NN \rowcolors \@@_rowcolors
1934 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1935 \cs_set_eq:NN \arraycolor \@@_arraycolor
1936 \cs_set_eq:NN \columncolor \@@_columncolor
1937 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1938 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1939 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1940 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1941 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1942 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1943 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1944 }
1945 \cs_new_protected:Npn \@@_exec_code_before:
1946 {

```

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...

```

1947 \clist_map_inline:Nn \l_@@_corners_cells_clist
1948 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1949 \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`.

```

1950 \@@_add_to_colors_seq:nn { { no color } } { }
1951 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1952 \group_begin:
1953 \@@_security_font_commands:

```

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

```

1954 \if_mode_math:
1955   \@@_exec_code_before_i:
1956 \else:

```

```

1957      $ % $
1958      \@@_exec_code_before_i:
1959      $ % $
1960      \fi:
1961      \group_end:
1962  }

```

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`).

```

1963 \cs_new_protected:Npn \@@_exec_code_before_i:
1964 {
1965   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1966   { \@@_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`.

```

1967   \exp_last_unbraced:No \@@_CodeBefore_keys:
1968   \g_@@_pre_code_before_tl

```

The following `\scan_stop` is a defensive programming for the case where the final user has written in the `\CodeBefore` something like `\rowcolor{red!15}` (without the second mandatory argument which should be a list of numbers of rows).

```

1969   \scan_stop:

```

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.

```

1970   \@@_actually_color:
1971   \l_@@_code_before_tl
1972   \q_stop
1973 }

```

```

1974 \keys_define:nn { nicematrix / CodeBefore }
1975 {
1976   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1977   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1978   sub-matrix .value_required:n = true ,
1979   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1980   delimiters / color .value_required:n = true ,
1981   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1982 }
1983 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1984 {
1985   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1986   \@@_CodeBefore:w
1987 }

```

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.

```

1988 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1989 {
1990   \bool_if:NTF \g_@@_aux_found_bool
1991   {
1992     \@@_pre_code_before:
1993     \legacy_if:nF { measuring@ } { #1 }
1994   }

```

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).

```

1995 {
1996   \pgfsys@markposition { \@@_env: - position }
1997   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1998   \pgfpicture
1999   \pgf@relevantforpicturesizefalse
2000   \endpgfpicture

```

The following picture corresponds to \@@_create_diag_nodes:

```

2001   \pgfpicture
2002   \pgfrememberpicturepositiononpagetrue
2003   \endpgfpicture

```

The following picture corresponds to \@@_create_blocks_nodes:.

```

2004   \pgfpicture
2005   \pgf@relevantforpicturesizefalse
2006   \pgfrememberpicturepositiononpagetrue
2007   \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

2008   \pgfpicture
2009   \pgf@relevantforpicturesizefalse
2010   \endpgfpicture
2011 }
2012 }

```

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.

```

2013 \cs_new_protected:Npn \@@_recreate_cell_nodes:
2014 {
2015   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2016   {
2017     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
2018     \pgfcoordinate { \@@_env: - row - ##1 - base }
2019     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2020     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2021     {
2022       \cs_if_exist:cT
2023       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
2024       {
2025         \pgfsys@getposition
2026         { \@@_env: - ##1 - #####1 - NW }
2027         \@@_node_position:
2028         \pgfsys@getposition
2029         { \@@_env: - ##1 - #####1 - SE }
2030         \@@_node_position_i:
2031         \@@_pgf_rect_node:nnn
2032         { \@@_env: - ##1 - #####1 }
2033         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2034         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2035       }
2036     }
2037   }
2038   \@@_create_extra_nodes:
2039   \@@_create_aliases_last:
2040 }

2041 \cs_new_protected:Npn \@@_create_aliases_last:
2042 {
2043   \int_step_inline:nn \c@iRow
2044   {
2045     \pgfnodealias
2046     { \@@_env: - ##1 - last }
2047     { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

2048     }
2049     \int_step_inline:nn \c@jCol
2050     {
2051         \pgfnodealias
2052         { \@@_env: - last - ##1 }
2053         { \@@_env: - \int_use:N \c@iRow - ##1 }
2054     }
2055     \pgfnodealias
2056     { \@@_env: - last - last }
2057     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2058 }

```

```

2059 \cs_new_protected:Npn \@@_create_blocks_nodes:
2060 {
2061     \pgfpicture
2062     \pgf@relevantforpicturesizefalse
2063     \pgfrememberpicturepositiononpagetrue
2064     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2065     { \@@_create_one_block_node:nnnnn ##1 }
2066     \endpgfpicture
2067 }

```

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.⁸

```

2068 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2069 {
2070     \tl_if_empty:nF { #5 }
2071     {
2072         \@@_qpoint:n { col - #2 }
2073         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2074         \@@_qpoint:n { #1 }
2075         \dim_set_eq:NN \l_tmpb_dim \pgf@y
2076         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2077         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2078         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2079         \@@_pgf_rect_node:nnnnn
2080         { \@@_env: - #5 }
2081         { \dim_use:N \l_tmpa_dim }
2082         { \dim_use:N \l_tmpb_dim }
2083         { \dim_use:N \l_@@_tmpc_dim }
2084         { \dim_use:N \pgf@y }
2085     }
2086 }

```

10 The environment `{NiceArrayWithDelims}`

```

2087 \NewDocumentEnvironment { NiceArrayWithDelims }
2088 { m m O { } m ! O { } t \CodeBefore }
2089 {
2090     \@@_provide_pgfsyspdfmark:
2091     \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

2092     \bgroup

2093     \tl_gset:Nn \g_@@_left_delim_tl { #1 }

```

⁸Moreover, 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).

```

2094 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2095 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2096 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2097 \int_gzero:N \g_@@_block_box_int
2098 \dim_gzero:N \g_@@_width_last_col_dim
2099 \dim_gzero:N \g_@@_width_first_col_dim
2100 \bool_gset_false:N \g_@@_row_of_col_done_bool
2101 \str_if_empty:NT \g_@@_name_env_str
2102 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2103 \bool_if:NTF \l_@@_tabular_bool
2104 \mode_leave_vertical:
2105 \@@_test_if_math_mode:
2106 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2107 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁹. 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.

```

2108 \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.

```

2109 \cs_if_exist:NT \tikz@library@external@loaded
2110 {
2111   \tikzexternaldisable
2112   \cs_if_exist:NT \ifstandalone
2113     { \tikzset { external / optimize = false } }
2114 }

```

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

```

2115 \int_gincr:N \g_@@_env_int
2116 \bool_if:NF \l_@@_block_auto_columns_width_bool
2117 { \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.

```

2118 \seq_gclear:N \g_@@_blocks_seq
2119 \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`.

```

2120 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2121 \seq_gclear:N \g_@@_pos_of_xdots_seq
2122 \tl_gclear_new:N \g_@@_code_before_tl
2123 \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.

```

2124 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2125 {
2126   \bool_gset_true:N \g_@@_aux_found_bool
2127   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2128 }
2129 { \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.

```

2130 \tl_gclear:N \g_@@_aux_tl

```

⁹e.g. `\color[rgb]{0.5,0.5,0}`

```

2131 \tl_if_empty:NF \g_@@_code_before_tl
2132 {
2133   \bool_set_true:N \l_@@_code_before_bool
2134   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2135 }
2136 \tl_if_empty:NF \g_@@_pre_code_before_tl
2137 { \bool_set_true:N \l_@@_code_before_bool }

```

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

```

2138 \bool_if:NTF \g_@@_delims_bool
2139 { \keys_set:nn { nicematrix / pNiceArray } }
2140 { \keys_set:nn { nicematrix / NiceArray } }
2141 { #3 , #5 }

2142 \@@_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:.

```

2143 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2144 }

```

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

```

2145 {
2146   \bool_if:NTF \l_@@_light_syntax_bool
2147   { \use:c { end @@-light-syntax } }
2148   { \use:c { end @@-normal-syntax } }
2149   $ % $
2150   \skip_horizontal:N \l_@@_right_margin_dim
2151   \skip_horizontal:N \l_@@_extra_right_margin_dim
2152   \hbox_set_end:
2153   \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.

```

2154 \bool_if:NT \l_@@_width_used_bool
2155 {
2156   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2157   { \@@_error_or_warning:n { width-without-X-columns } }
2158 }

```

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 .

```

2159 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2160 \@@_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).

```

2161 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2162 {
2163   \bool_if:NF \l_@@_last_row_without_value_bool
2164   {
2165     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2166     {
2167       \@@_error_or_warning:n { Wrong-last-row }
2168       \int_set_eq:NN \l_@@_last_row_int \c_iRow

```

```

2169         }
2170     }
2171 }

```

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”.¹⁰

```

2172 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2173 \bool_if:NTF \g_@@_last_col_found_bool
2174 { \int_gdecr:N \c@jCol }
2175 {
2176     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2177     { \@@_error:n { last~col~not~used } }
2178 }

```

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

```

2179 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2180 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2181 { \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. 98).

```

2182 \int_if_zero:nT \l_@@_first_col_int
2183 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2184 \bool_if:nTF { ! \g_@@_delims_bool }
2185 {
2186     \str_if_eq:eeTF c \l_@@_baseline_tl
2187     \@@_use_arraybox_with_notes_c:
2188     {
2189         \str_if_eq:eeTF b \l_@@_baseline_tl
2190         \@@_use_arraybox_with_notes_b:
2191         \@@_use_arraybox_with_notes:
2192     }
2193 }

```

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).

```

2194 {
2195     \int_if_zero:nTF \l_@@_first_row_int
2196     {
2197         \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2198         \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2199     }
2200     { \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”.¹¹

```

2201 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2202 {
2203     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2204     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2205 }
2206 { \dim_zero:N \l_tmpb_dim }
2207 \hbox_set:Nn \l_tmpa_box
2208 {
2209     \m@th
2210     $ % $
2211     \@@_color:o \l_@@_delimiters_color_tl

```

¹⁰We remind that the potential “first column” (exterior) has the number 0.

¹¹A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

2212         \exp_after:wN \left \g_@@_left_delim_tl
2213         \vcenter
2214         {

```

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.

```

2215         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2216         \hbox
2217         {
2218             % \bool_if:NTF \l_@@_tabular_bool
2219             % { \skip_horizontal:n { - \tabcolsep } }
2220             % { \skip_horizontal:n { - \arraycolsep } }
2221             \skip_horizontal:n
2222             { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2223             \@@_use_arraybox_with_notes_c:
2224             \skip_horizontal:n
2225             { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2226             % \bool_if:NTF \l_@@_tabular_bool
2227             % { \skip_horizontal:n { - \tabcolsep } }
2228             % { \skip_horizontal:n { - \arraycolsep } }
2229         }

```

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

```

2230         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2231     }
2232     \exp_after:wN \right \g_@@_right_delim_tl
2233     $ % $
2234 }

```

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.

```

2235     \bool_if:NTF \l_@@_delimiters_max_width_bool
2236     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2237     \@@_put_box_in_flow:
2238 }

```

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. 99).

```

2239     \bool_if:NT \g_@@_last_col_found_bool
2240     { \skip_horizontal:N \g_@@_width_last_col_dim }
2241     \@@_test_false_columns:
2242     \bool_if:NT \l_@@_preamble_bool
2243     {
2244         \int_compare:nNt \c@jCol < \g_@@_static_num_of_col_int
2245         { \@@_err_columns_not_used: }
2246     }
2247     \@@_after_array:

```

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

```

2248     \egroup

```

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

```

2249     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2250     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2251     \iow_now:Ne \@mainaux
2252     {
2253         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2254         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2255         { \exp_not:o \g_@@_aux_tl }
2256     }
2257     \iow_now:Nn \@mainaux { \ExplSyntaxOff }
2258     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }

```

```
2259 }
```

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

```
2260 \cs_new_protected:Npn \@@_err_columns_not_used:
2261 {
2262   \@@_warning:n { columns-not-used }
2263   \cs_gset:Npn \@@_err_columns_not_used: { }
2264 }
```

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*.

```
2265 \cs_new_protected:Npn \@@_compute_width_X:
2266 {
2267   \tl_gput_right:Ne \g_@@_aux_tl
2268   {
2269     \bool_set_true:N \l_@@_X_columns_aux_bool
2270     \dim_set:Nn \l_@@_X_columns_dim
2271     {
```

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`

```
2272     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2273     { \dim_use:N \l_@@_X_columns_dim }
2274     {
2275       \dim_compare:nNnTF
2276       {
2277         \dim_abs:n
2278         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2279       }
2280       <
2281       { 0.001 pt }
2282       { \dim_use:N \l_@@_X_columns_dim }
2283       {
2284         \dim_eval:n
2285         {
2286           \l_@@_X_columns_dim
2287           +
2288           \fp_to_dim:n
2289           {
2290             (
2291               \dim_eval:n
2292               { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2293             )
2294             / \fp_use:N \g_@@_total_X_weight_fp
2295           }
2296         }
2297       }
2298     }
2299   }
2300 }
2301 }
```

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 `\l_@@_array_preamble_tl`.

```
2302 \cs_new_protected:Npn \@@_transform_preamble:
2303 {
```

First, some tuning.

```
2304 \@@_transform_preamble_i:
```

We will now actually make the preamble. The preamble provided by the final user (when he uses an environment with preamble, such as `{NiceTabular}`, etc.) has been stored in `\g_@@_user_preamble_tl`. Using that information, we will create `\l_@@_array_preamble_tl`, which will be the preamble that we will provide to `{array}`.

```
2305 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
```

Now, some post-treatment.

```
2306 \@@_transform_preamble_ii:
2307 }
```

```
2308 \cs_new_protected:Npn \@@_transform_preamble_i:
2309 {
```

The following counter will be used during the construction of `\l_@@_array_preamble_tl`, and, after that construction, it will be the number of columns of the tabular as indicated by the user preamble (for the environments with preamble).

```
2310 \int_gzero:N \g_@@_static_num_of_col_int
```

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

```
2311 \seq_gclear:N \g_@@_cols_vlism_seq
2312 \clist_gclear:N \g_@@_false_cols_clist
2313 \clist_gclear:N \g_@@_false_rows_clist
```

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

```
2314 \int_zero:N \l_tmpa_int
2315 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2316 {
2317   \tl_set:Nn \l_@@_array_preamble_tl
2318     { ! { \skip_horizontal:N \arrayrulewidth } }
2319 }
2320 {
2321   \clist_if_in:NnT \l_@@_vlines_clist 1
2322   {
2323     \tl_set:Nn \l_@@_array_preamble_tl
2324       { ! { \skip_horizontal:N \arrayrulewidth } }
2325   }
2326 }
2327 }
```

```
2328 \cs_new_protected:Npn \@@_transform_preamble_ii:
2329 {
2330   \@@_replace_columncolor:
```

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

```
2331 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2332 {
2333   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2334   { \bool_gset_true:N \g_@@_delims_bool }
2335 }
2336 { \bool_gset_true:N \g_@@_delims_bool }
```

We complete the preamble with the potential “exterior columns” (on both sides).

```

2337 \int_if_zero:nTF \l_@@_first_col_int
2338 { \tl_put_left:No \l_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2339 {
2340   \bool_if:NF \g_@@_delims_bool
2341   {
2342     \bool_if:NF \l_@@_tabular_bool
2343     {
2344       \clist_if_empty:NT \l_@@_vlines_clist
2345       {
2346         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2347         { \tl_put_left:Nn \l_@@_array_preamble_tl { @ { } } }
2348       }
2349     }
2350   }
2351 }
2352 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2353 { \tl_put_right:No \l_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2354 {
2355   \bool_if:NF \g_@@_delims_bool
2356   {
2357     \bool_if:NF \l_@@_tabular_bool
2358     {
2359       \clist_if_empty:NT \l_@@_vlines_clist
2360       {
2361         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2362         { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { } } }
2363       }
2364     }
2365   }
2366 }

```

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.

```

2367 \tag_if_active:F
2368 {

```

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

```

2369 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2370 {
2371   \tl_put_right:Nn \l_@@_array_preamble_tl
2372   { > { \@@_err_too_many_cols: } 1 }
2373 }
2374 }
2375 }

```

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.

```

2376 \cs_new_protected:Npn \@@_rec_preamble:n #1
2377 {

```

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.¹²

```

2378 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2379 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2380 {

```

Now, the columns defined by `\newcolumn` type of array.

```

2381 \cs_if_exist:cTF { NC @ find @ #1 }
2382 {
2383 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2384 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2385 }
2386 {
2387 \str_case:nnF { #1 }
2388 {
2389 { S } { \@@_fatal:n { unknown~column~type~S } }
2390 { ; } { \@@_fatal:n { unknown~column~type~; } }
2391 { R } { \@@_fatal:nn { unknown~column~type~RCL } { #1 } }
2392 { C } { \@@_fatal:nn { unknown~column~type~RCL } { #1 } }
2393 { L } { \@@_fatal:nn { unknown~column~type~RCL } { #1 } }
2394 }
2395 { \@@_fatal:nn { unknown~column~type } { #1 } }
2396 }
2397 }
2398 }

2399 \cs_new_protected:cpn { @@ _ > : } #1 #2
2400 {
2401 \tl_put_right:Nn \l_@@_pre_cell_tl { > { #2 } }
2402 \@@_rec_preamble:n
2403 }

2404 \cs_new_protected:Npn \@@_use_pre_cell_tl:
2405 {
2406 \tl_put_right:No \l_@@_array_preamble_tl \l_@@_pre_cell_tl
2407 \tl_clear:N \l_@@_pre_cell_tl
2408 }

```

For c, l and r

```

2409 \cs_new_protected:Npn \@@_c: #1
2410 {
2411 \@@_use_pre_cell_tl:
2412 \tl_put_right:Nn \l_@@_array_preamble_tl
2413 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2414 \@@_rec_preamble_after_col:n
2415 }

2416 \cs_new_protected:Npn \@@_l: #1
2417 {
2418 \@@_use_pre_cell_tl:
2419 \tl_put_right:Nn \l_@@_array_preamble_tl
2420 {
2421 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2422 l
2423 < \@@_cell_end:
2424 }

```

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

```

2425 \@@_rec_preamble_after_col:n
2426 }

```

¹²We do that because it's an easy way to insert the letter at some places in the code that we will add to `\l_@@_array_preamble_tl`.

```

2427 \cs_new_protected:Npn \@@_r: #1
2428 {
2429   \@@_use_pre_cell_tl:
2430   \tl_put_right:Nn \l_@@_array_preamble_tl
2431   {
2432     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2433     r
2434     < \@@_cell_end:
2435   }

```

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

```

2436   \@@_rec_preamble_after_col:n
2437 }

```

For ! and @

```

2438 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2439 {
2440   \tl_put_right:Nn \l_@@_array_preamble_tl { #1 { #2 } }
2441   \@@_rec_preamble:n
2442 }
2443 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2444 \cs_new_protected:cpn { @@ _ | : } #1
2445 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2446   \int_incr:N \l_tmpa_int
2447   \@@_make_preamble_i_i:n
2448 }
2449 \cs_new_protected:Npn \@@_F: #1
2450 {
2451   \tl_put_right:Nn \l_@@_array_preamble_tl
2452   {
2453     > {
2454       \skip_horizontal:n { -2 \col@sep + \l_@@_width_of_false_dim }
2455       \skip_horizontal:N \c_zero_dim
2456       \@@_cell_begin:
2457     }
2458     c
2459     < {
2460       \@@_cell_end:
2461     }
2462   }

```

The clist \g_@@_false_cols_clist is the list of the occurrences of the letter F (for the “false columns”).

```

2463   \clist_gput_right:Ne \g_@@_false_cols_clist
2464   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2465   \socket_assign_plug:nn { nicematrix / horizontal-space } { with-false-col }

```

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

```

2466   \@@_rec_preamble_after_col:n
2467 }
2468 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2469 {

```

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

```

2470   \str_if_eq:nnTF { #1 } { | }
2471   { \use:c { @@ _ | : } | }
2472   { \@@_make_preamble_i_ii:nn { } #1 }
2473 }

```

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

```

2474 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2475 {
2476   \str_if_eq:nnTF { #2 } { [ ]
2477     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2478       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2479     }
2480 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2481 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2482 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2483 {
2484   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2485   \tl_put_right:Ne \l_@@_array_preamble_tl
2486   {

```

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

```

2487   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2488   }
2489   \tl_gput_right:Ne \g_@@_rules_tl
2490   {

```

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

```

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

```

We will use a version slightly more efficient:

```

2491 {
2492   \int_compare:nNnT \l_tmpa_int > 1
2493   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2494   \int_set:Nn \l_@@_position_int
2495   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2496   \dim_set:Nn \l_@@_rule_width_after_dim
2497   { \dim_use:N \l_@@_rule_width_before_dim }
2498   \@@_draw_vrule:n { #2 }
2499 }

```

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.

```

2500   }
2501   \int_zero:N \l_tmpa_int
2502   \str_if_eq:nnT { #1 } { \s_stop }
2503   { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
2504   \@@_rec_preamble:n #1
2505 }

```

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.

```

2506 \keys_define:nn { nicematrix / p-column }
2507 {
2508   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2509   r .value_forbidden:n = true ,
2510   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2511   c .value_forbidden:n = true ,
2512   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,

```

```

2513   l .value_forbidden:n = true ,
2514   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2515   S .value_forbidden:n = true ,
2516   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2517   p .value_forbidden:n = true ,
2518   t .meta:n = p ,
2519   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2520   m .value_forbidden:n = true ,
2521   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2522   b .value_forbidden:n = true
2523 }

```

For p but also b and m.

```

2524 \cs_new_protected:Npn \@@_p: #1
2525 {
2526   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2527   \@@_make_preamble_ii_i:n
2528 }
2529 \cs_new_eq:NN \@@_b: \@@_p:
2530 \cs_new_eq:NN \@@_m: \@@_p:

2531 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2532 {
2533   \str_if_eq:nnTF { #1 } { [ ]
2534     { \@@_make_preamble_ii_ii:w [ ]
2535       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2536     }
2537   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2538     { \@@_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.

```

2539 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2540 {

```

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).

```

2541   \str_set:Nn \l_@@_hpos_col_str { j }
2542   \@@_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`.

```

2543   \setlength { \l_tmpa_dim } { #2 }
2544   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2545 }

2546 \cs_new_protected:Npn \@@_keys_p_column:n #1
2547 { \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.

```

2548 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2549 {

```

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

```

2550   \use:e
2551   {
2552     \@@_make_preamble_ii_vi:nnnnnnnn
2553     { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }

```

```

2554 { #1 }
2555 {
2556 \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.

```

2557 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2558 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2559 {

```

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

```

2560 \def \exp_not:N \l_@@_hpos_cell_tl
2561 { \str_lowercase:f { \l_@@_hpos_col_str } }
2562 }
2563 \IfPackageLoadedTF { ragged2e }
2564 {
2565 \str_case:on \l_@@_hpos_col_str
2566 {

```

The following `\exp_not:N` are mandatory.

```

2567 c { \exp_not:N \Centering }
2568 l { \exp_not:N \RaggedRight }
2569 r { \exp_not:N \RaggedLeft }
2570 }
2571 }
2572 {
2573 \str_case:on \l_@@_hpos_col_str
2574 {
2575 c { \exp_not:N \centering }
2576 l { \exp_not:N \raggedright }
2577 r { \exp_not:N \raggedleft }
2578 }
2579 }
2580 #3
2581 }
2582 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2583 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2584 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2585 { #2 }
2586 {
2587 \str_case:onF \l_@@_hpos_col_str
2588 {
2589 { j } { c }
2590 { si } { c }
2591 }

```

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

```

2592 { \str_lowercase:f \l_@@_hpos_col_str }
2593 }
2594 }

```

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

```

2595 \@@_rec_preamble_after_col:n
2596 }

```

#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*.

```

2597 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2598 {
2599   % \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2600   % {
2601   %   \tl_put_right:Nn \l_@@_array_preamble_tl
2602   %     { > \@@_test_if_empty_for_S: }
2603   % }
2604   % {
2605   %   \str_if_eq:eeTF { #7 } { varwidth }
2606   %   {
2607   %     \tl_put_right:Nn \l_@@_array_preamble_tl
2608   %       { > \@@_test_if_empty_varwidth: }
2609   %   }
2610   %   { \tl_put_right:Nn \l_@@_array_preamble_tl { > \@@_test_if_empty: } }
2611   % }
2612   \tl_put_right:Ne \l_@@_array_preamble_tl
2613   {
2614   >
2615   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2616   { \@@_test_if_empty_for_S: }
2617   {
2618     \str_if_eq:eeTF { #7 } { varwidth }
2619     \@@_test_if_empty_varwidth:
2620     \@@_test_if_empty:
2621   }
2622   }
2623   \@@_use_pre_cell_tl:
2624   \tl_put_right:Nn \l_@@_array_preamble_tl
2625   {
2626   > {

```

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.

```

2627   \dim_set:Nn \l_@@_col_width_dim { #2 }
2628   \@@_cell_begin:

```

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

```

2629   \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty` (version 2.7b).

```

2630   \everypar
2631   {
2632     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2633     \everypar { }
2634   }

```

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

```

2635   #3

```

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

```

2636   \g_@@_row_style_tl
2637   \arraybackslash
2638   #5
2639   }
2640   #8
2641   < {
2642   #6

```

The following line has been taken from `array.sty` (version 2.7b).

```
2643         \finalstrut \@arstrutbox
2644         \use:c { end #7 }
```

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

```
2645         #4
2646         \@@_cell_end:
2647     }
2648 }
2649 }
```

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

```
2650 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2651 {
```

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 `&`).

```
2652     \group_align_safe_begin:
2653     \peek_meaning:NTF &
2654     \@@_the_cell_is_empty: % contains \group_align_safe_end:
2655     {
2656         \peek_meaning:NTF \\\
2657         \@@_the_cell_is_empty: % idem
2658         {
2659             \peek_meaning:NTF \crcr
2660             \@@_the_cell_is_empty: % idem
2661             \group_align_safe_end:
2662         }
2663     }
2664 }
```

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

```
2665 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2666 {
2667     \group_align_safe_begin:
2668     \peek_meaning:NTF &
2669     \@@_the_cell_is_empty_varwidth:
2670     {
2671         \peek_meaning:NTF \\\
2672         \@@_the_cell_is_empty_varwidth:
2673         {
2674             \peek_meaning:NTF \crcr
2675             \@@_the_cell_is_empty_varwidth:
2676             \group_align_safe_end:
2677         }
2678     }
2679 }

2680 \cs_new_protected:Npn \@@_the_cell_is_empty:
2681 {
2682     \group_align_safe_end:
2683     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2684     {
```

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

```
2685         \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`).

```
2686         \skip_horizontal:N \l_@@_col_width_dim
2687     }
2688 }
```

```

2689 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2690 {
2691   \group_align_safe_end:
2692   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2693     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2694 }
2695 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2696 {
2697   \peek_meaning:NT \__siunitx_table_skip:n
2698     { \bool_gset_true:N \g_@@_empty_cell_bool }
2699 }

```

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.

```

2700 \cs_new_protected:Npn \@@_center_cell_box:
2701 {

```

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

```

2702   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2703     {
2704       \dim_compare:nNnT
2705         { \box_ht:N \l_@@_cell_box }
2706         >

```

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).

```

2707       { \box_ht:N \strutbox }
2708       {
2709         \hbox_set:Nn \l_@@_cell_box
2710           {
2711             \box_move_down:nn
2712               {
2713                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2714                   + \baselineskip ) / 2
2715               }
2716             { \box_use:N \l_@@_cell_box }
2717           }
2718       }
2719     }
2720 }

```

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

```

2721 \cs_new_protected:Npn \@@_V: #1 #2
2722 {
2723   \str_if_eq:nnTF { #2 } { [ ] }
2724     { \@@_make_preamble_V_i:w [ ] }
2725     { \@@_make_preamble_V_i:w [ ] { #2 } }
2726 }
2727 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2728 { \@@_make_preamble_V_ii:nn { #1 } }
2729 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2730 {
2731   \str_set:Nn \l_@@_vpos_col_str { p }
2732   \str_set:Nn \l_@@_hpos_col_str { j }
2733   \@@_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`.

```

2734 \setlength { \l_tmpa_dim } { #2 }
2735 \IfPackageLoadedTF { varwidth }
2736 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2737 {
2738   \@@_error_or_warning:n { varwidth-not-loaded }
2739   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2740 }
2741 }
2742 % \end{macrocod}
2743 %
2744 % \medskip
2745 % For |w| and |W|
2746 % \begin{macrocode}
2747 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2748 \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.

```

2749 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2750 {
2751   \tl_if_in:nnTF { clrs } { #3 }
2752   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2753   {
2754     \@@_error:nn { Invalid-argument-for-w } { #3 }
2755     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2756   }
2757 }
2758 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2759 {
2760   \str_if_eq:nnTF { #3 } { s }
2761   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2762   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2763 }

```

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.

```

2764 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2765 {
2766   \@@_use_pre_cell_tl:
2767   \tl_put_right:Nn \l_@@_array_preamble_tl
2768   {
2769     > {

```

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`.

```

2770 \setlength { \l_@@_col_width_dim } { #2 }
2771 \@@_cell_begin:
2772 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2773 }
2774 c
2775 < {
2776   \@@_cell_end_for_w_s:
2777   #1
2778   \@@_adjust_size_box:

```

```

2779         % \box_use_drop:N \l_@@_cell_box
2780     }
2781 }

```

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

```

2782 \@@_rec_preamble_after_col:n
2783 }

```

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

```

2784 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2785 {
2786     \@@_use_pre_cell_tl:

```

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`.

We don't want to compute the width of the column with `\setlength` in each cell of the column. That's why we compute it now and use an expansion after.

We could do the same thing previously for a column `w` of type `s` but the type `s` is almost never used...

```

2787     \setlength { \l_tmpa_dim } { #4 }
2788     \tl_put_right:Ne \l_@@_array_preamble_tl
2789     {
2790         > {

```

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 need `\dim_use:N` because we are in an expansion.

```

2791         \dim_set:Nn \l_@@_col_width_dim { \dim_use:N \l_tmpa_dim }
2792         \hbox_set:Nw \l_@@_cell_box
2793         \@@_cell_begin:
2794         \tl_set:Nn \exp_not:N \l_@@_hpos_cell_tl { #3 }
2795     }
2796 }
2797 \tl_put_right:Nn \l_@@_array_preamble_tl
2798 {
2799     c
2800     < {
2801         \@@_cell_end:
2802         \hbox_set_end:
2803         #1
2804         \@@_adjust_size_box:
2805         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2806     }
2807 }

```

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

```

2808 \@@_rec_preamble_after_col:n
2809 }

```

```

2810 \cs_new_protected:Npn \@@_special_W:
2811 {
2812     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2813     { \@@_warning:n { W-warning } }
2814 }

```

For `S` (of `siunitx`).

```

2815 \AddToHook { package / siunitx / after }
2816 {
2817     \cs_new_protected:Npn \@@_S: #1 #2
2818     {
2819         \str_if_eq:nnTF { #2 } { [ ] }
2820         { \@@_make_preamble_S:w [ ] }
2821         { \@@_make_preamble_S:w [ ] { #2 } }

```

```

2822     }
2823 }
2824 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2825 { \@@_make_preamble_S_i:n { #1 } }
2826 \cs_new_protected:Npn \@@_test_siunitx:
2827 {
2828   \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2829   { \@@_fatal:n { siunitx~too~old } }
2830   \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2831 }
2832 % \end{macrocode}
2833 %
2834 % \begin{macrocode}
2835 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2836 {
2837   \@@_test_siunitx:
2838   \@@_use_pre_cell_tl:
2839   \tl_put_right:Nn \l_@@_array_preamble_tl
2840   {
2841     > {

```

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).

```

2842       \socket_assign_plug:n { nicematrix / siunitx-wrap } { active }
2843       \keys_set:n { siunitx } { #1 }
2844       \@@_cell_begin:
2845       \siunitx_cell_begin:w
2846     }
2847     c
2848     <
2849     {
2850       \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, it will stay local within the cell of the underlying `\halign`).

```

2851       \tl_gput_right:N \g_@@_cell_after_hook_tl
2852       {
2853         \bool_if:NTF \l__siunitx_table_text_bool
2854         \bool_set_true:N
2855         \bool_set_false:N
2856         \l__siunitx_table_text_bool
2857       }
2858       \@@_cell_end:
2859     }
2860 }

```

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

```

2861   \@@_rec_preamble_after_col:n
2862 }

```

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

```

2863 \cs_new_protected:cpn { @@_ _ \token_to_str:N ( : } #1 #2
2864 {
2865   \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.

```

2866   \int_if_zero:nTF \g_@@_static_num_of_col_int
2867   {
2868     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2869     {

```

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

```

2870         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2871         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2872         \@@_rec_preamble:n #2
2873     }
2874     {
2875         \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2876         \@@_make_preamble_iv:nn { #1 } { #2 }
2877     }
2878 }
2879 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2880 }
2881 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2882 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2883 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2884 {
2885     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2886     {
2887         \@@_delimiter:nnn #1
2888         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2889         \c_true_bool
2890     }
2891     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2892     {
2893         \@@_error:nn { delimiter~after~opening } { #2 }
2894         \@@_rec_preamble:n
2895     }
2896     { \@@_rec_preamble:n #2 }
2897 }

```

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

```

2898 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2899 { \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}`).

```

2900 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2901 {
2902     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2903     \tl_if_in:nnTF { ) ] \} } { #2 }
2904     { \@@_make_preamble_v:nnn #1 #2 }
2905     {
2906         \str_if_eq:nnTF \s_stop { #2 }
2907         {
2908             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2909             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2910             {
2911                 \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2912                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2913                 {
2914                     \@@_delimiter:nnn #1
2915                     { \int_use:N \g_@@_static_num_of_col_int }
2916                     \c_false_bool
2917                 }
2918                 \@@_rec_preamble:n #2
2919             }
2920         }
2921         {
2922             \tl_if_in:nnT { ( [ \{ \left } { #2 }

```

```

2923         { \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } } }
2924     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2925     {
2926         \@@_delimiter:nnn #1
2927         { \int_use:N \g_@@_static_num_of_col_int }
2928         \c_false_bool
2929     }
2930     \@@_rec_preamble:n #2
2931 }
2932 }
2933 }
2934 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2935 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N } : }
2936 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2937 {
2938     \str_if_eq:nnTF \s_stop { #3 }
2939     {
2940         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2941         {
2942             \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2943             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2944             {
2945                 \@@_delimiter:nnn #1
2946                 { \int_use:N \g_@@_static_num_of_col_int }
2947                 \c_false_bool
2948             }
2949             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2950         }
2951         {
2952             \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2953             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2954             {
2955                 \@@_delimiter:nnn #1
2956                 { \int_use:N \g_@@_static_num_of_col_int }
2957                 \c_false_bool
2958             }
2959             \@@_error:nn { double~closing~delimiter } { #2 }
2960         }
2961     }
2962     {
2963         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2964         {
2965             \@@_delimiter:nnn #1
2966             { \int_use:N \g_@@_static_num_of_col_int }
2967             \c_false_bool
2968         }
2969         \@@_error:nn { double~closing~delimiter } { #2 }
2970         \@@_rec_preamble:n #3
2971     }
2972 }
2973 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2974 { \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 @{...}.

```

2975 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2976 {
2977     % \int_gincr:N \g_@@_static_num_of_col_int
2978     \str_if_eq:nnTF { #1 } { < }
2979     { \@@_rec_preamble_after_col_i:n }

```

```

2980 {
2981   \int_gincr:N \g_@@_static_num_of_col_int % modified 2026-09-09
2982   \str_if_eq:nnTF { #1 } { @ }
2983   { \@@_rec_preamble_after_col_ii:n }
2984   {
2985     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2986     {
2987       \tl_put_right:Nn \l_@@_array_preamble_tl
2988         { ! { \skip_horizontal:N \arrayrulewidth } }
2989     }
2990     {
2991       \clist_if_in:NeT \l_@@_vlines_clist
2992       { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2993       {
2994         \tl_put_right:Nn \l_@@_array_preamble_tl
2995           { ! { \skip_horizontal:N \arrayrulewidth } }
2996       }
2997     }
2998     \@@_rec_preamble:n { #1 }
2999   }
3000 }
3001 }
3002 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
3003 {
3004   \tl_put_right:Nn \l_@@_array_preamble_tl { < { #1 } }
3005   \@@_rec_preamble_after_col:n
3006 }

```

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.

```

3007 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
3008 {
3009   \str_if_eq:eeTF \l_@@_vlines_clist { all }
3010   {
3011     \tl_put_right:Nn \l_@@_array_preamble_tl
3012       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3013   }
3014   {
3015     \clist_if_in:NeTF \l_@@_vlines_clist
3016     { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3017     {
3018       \tl_put_right:Nn \l_@@_array_preamble_tl
3019         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3020     }
3021     { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { #1 } } }
3022   }
3023   \@@_rec_preamble:n
3024 }

```

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.

```

3025 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
3026 { \@@_rec_preamble:n }
3027 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
3028 {
3029   \tl_clear:N \l_tmpa_tl

```

We use the neutral token \NC@find as a kind of \relax to stop the analysis after each iteration of the pattern.

```

3030   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 \NC@find } }
3031   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
3032 }

```

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.

```

3033 \cs_new_protected:Npn \@@_X: #1 #2
3034 {
3035   \str_if_eq:nnTF { #2 } { [ ] }
3036   { \@@_make_preamble_X:w [ ] }
3037   { \@@_make_preamble_X:w [ ] #2 }
3038 }
3039 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
3040 { \@@_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`.

```

3041 \keys_define:nn { nicematrix / X-column }
3042 {
3043   V .code:n =
3044   \IfPackageLoadedTF { varwidth }
3045   {
3046     \bool_set_true:N \l_@@_V_of_X_bool
3047     \bool_gset_true:N \g_@@_V_of_X_bool
3048   }
3049   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
3050   unknown .code:n =
3051   \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
3052   { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
3053   { \@@_error_or_warning:n { invalid~weight } }
3054 }

```

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

```

3055 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
3056 {

```

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).

```

3057   \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).

```

3058   \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`).

```

3059   \fp_set:Nn \l_tmpa_fp { 1.0 }
3060   \@@_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.

```

3061   \bool_set_false:N \l_@@_V_of_X_bool
3062   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

3063   \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).

```

3064   \bool_if:NTF \l_@@_X_columns_aux_bool
3065   {
3066     \@@_make_preamble_ii_iv:nnn

```

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

```

3067         { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
3068         { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
3069         { \@@_no_update_width: }
3070     }

```

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.

```

3071     {
3072         \tl_put_right:Nn \l_@@_array_preamble_tl
3073         {
3074             > {
3075                 \@@_cell_begin:
3076                 \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`.

```

3077         \NotEmpty

```

The following code will nullify the box of the cell.

```

3078         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3079         { \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`.

```

3080         \begin { minipage } { 5 cm } \arraybackslash
3081     }
3082     c
3083     < {
3084         \end { minipage }
3085         \@@_cell_end:
3086     }
3087 }
3088 \@@_rec_preamble_after_col:n
3089 }
3090 }

```

```

3091 \cs_new_protected:Npn \@@_no_update_width:
3092 {
3093     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3094     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3095 }

```

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

```

3096 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3097 {
3098     \seq_gput_right:Ne \g_@@_cols_vlism_seq
3099     { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3100     \tl_put_right:Ne \l_@@_array_preamble_tl
3101     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3102     \@@_rec_preamble:n
3103 }

```

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.

```

3104 \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).

```

3105 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3106 { \@@_fatal:n { Preamble~forgotten } }
3107 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3108 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3109 { @@ _ \token_to_str:N \hline : }
3110 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3111 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
3112 { @@ _ \token_to_str:N \hline : }
3113 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3114 { @@ _ \token_to_str:N \hline : }
3115 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3116 { @@ _ \token_to_str:N \hline : }
3117 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3118 { @@ _ \token_to_str:N \hline : }
3119 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3120 { \@@_fatal:n { NiceTabularX~probably~required } }
3121 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3122 { @@ _ \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).

```

3123 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3124 {

```

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`.

```

3125 \multispan { #1 }
3126 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3127 \begingroup
3128 \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3129 \tl_gclear:N \g_@@_preamble_tl
3130 \@@_make_m_preamble:n #2 \q_stop

```

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

```

3131 \def \@addamp
3132 {
3133 \legacy_if:nTF { @firstamp }
3134 { \legacy_if_set_false:n { @firstamp } }
3135 { \@preamerr 5 }
3136 }
3137 \exp_args:No \mkpream \g_@@_preamble_tl
3138 \@addtopreamble \empty
3139 \endgroup
3140 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3141 \int_compare:nNnT { #1 } > 1
3142 {
3143 \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3144 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3145 \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }

```

```

3146 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3147 {
3148   {
3149     \int_if_zero:nTF \c@jCol
3150     { \int_eval:n { \c@iRow + 1 } }
3151     { \int_use:N \c@iRow }
3152   }
3153   { \int_eval:n { \c@jCol + 1 } }
3154   {
3155     \int_if_zero:nTF \c@jCol
3156     { \int_eval:n { \c@iRow + 1 } }
3157     { \int_use:N \c@iRow }
3158   }
3159   { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3160   { }
3161 }
3162 }

```

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

```

3163 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3164 {
3165   \tl_gput_right:Ne \g_@@_pre_code_before_tl
3166   {
3167     \@@_rectanglecolor [ ##1 ]
3168     { \exp_not:n { ##2 } }
3169     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3170     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3171   }
3172   \ignorespaces
3173 }

```

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

```

3174 \def \@sharp { #3 }
3175 \@arstrut
3176 \@preamble
3177 \null

```

We add some lines.

```

3178 \int_gadd:Nn \c@jCol { #1 - 1 }
3179 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3180 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3181 \ignorespaces
3182 }

```

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`.

```

3183 \
3184 \cs_new_protected:Npn \@_make_m_preamble:n #1
3185 {
3186   \str_case:nnF { #1 }
3187   {
3188     c { \@_make_m_preamble_i:n #1 }
3189     l { \@_make_m_preamble_i:n #1 }
3190     X { \@_make_m_preamble_i:n l }
3191     r { \@_make_m_preamble_i:n #1 }
3192     > { \@_make_m_preamble_ii:nn #1 }
3193     ! { \@_make_m_preamble_ii:nn #1 }
3194     @ { \@_make_m_preamble_ii:nn #1 }
3195     | { \@_make_m_preamble_iii:n #1 }

```

```

3196 p { \@@_make_m_preamble_iv:nnn t #1 }
3197 m { \@@_make_m_preamble_iv:nnn c #1 }
3198 b { \@@_make_m_preamble_iv:nnn b #1 }
3199 w { \@@_make_m_preamble_v:nnnn { } #1 }
3200 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3201 \q_stop { }
3202 }
3203 {
3204 \cs_if_exist:cTF { NC @ find @ #1 }
3205 {
3206 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3207 \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3208 }
3209 {
3210 \str_if_eq:nnTF { #1 } { S }
3211 { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3212 { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3213 }
3214 }
3215 }

```

For c, l and r

```

3216 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3217 {
3218 \tl_gput_right:Nn \g_@@_preamble_tl
3219 {
3220 > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3221 #1
3222 < \@@_cell_end:
3223 }

```

We test for the presence of a <.

```

3224 \@@_make_m_preamble_x:n
3225 }

```

For >, ! and @

```

3226 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3227 {
3228 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3229 \@@_make_m_preamble:n
3230 }

```

For |

```

3231 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3232 {
3233 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3234 \@@_make_m_preamble:n
3235 }

```

For p, m and b

```

3236 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3237 {
3238 \tl_gput_right:Nn \g_@@_preamble_tl
3239 {
3240 > {
3241 \@@_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`.

```

3242 \setlength { \l_tmpa_dim } { #3 }
3243 \begin { minipage } [ #1 ] { \l_tmpa_dim }
3244 \mode_leave_vertical:

```

```

3245         \arraybackslash
3246         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3247     }
3248     c
3249     < {
3250         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3251         \end { minipage }
3252         \@@_cell_end:
3253     }
3254 }

```

We test for the presence of a <.

```

3255     \@@_make_m_preamble_x:n
3256 }

```

For w and W

```

3257 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3258 {
3259     \tl_gput_right:Nn \g_@@_preamble_tl
3260     {
3261         > {
3262             \dim_set:Nn \l_@@_col_width_dim { #4 }
3263             \hbox_set:Nw \l_@@_cell_box
3264             \@@_cell_begin:
3265             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3266         }
3267         c
3268         < {
3269             \@@_cell_end:
3270             \hbox_set_end:
3271             \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3272             #1
3273             \@@_adjust_size_box:
3274             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3275         }
3276     }

```

We test for the presence of a <.

```

3277     \@@_make_m_preamble_x:n
3278 }

```

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

```

3279 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3280 {
3281     \str_if_eq:nnTF { #1 } { < }
3282     \@@_make_m_preamble_ix:n
3283     { \@@_make_m_preamble:n { #1 } }
3284 }
3285 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3286 {
3287     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3288     \@@_make_m_preamble_x:n
3289 }

```

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).

```

3290 \cs_new_protected:Npn \@@_put_box_in_flow:
3291 {
3292     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3293     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3294     \str_if_eq:eeTF \l_@@_baseline_tl { c }

```

```

3295 { \box_use_drop:N \l_tmpa_box }
3296 \@@_put_box_in_flow_i:
3297 }

```

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

```

3298 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3299 {
3300   \pgfpicture
3301     \@@_qpoint:n { row - 1 }
3302     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3303     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3304     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3305     \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).

```

3306   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3307   {
3308     \int_set:Nn \l_tmpa_int
3309       { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3310     \bool_lazy_or:nnT
3311       { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3312       { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3313     {
3314       \@@_error:n { bad-value-for-baseline-line }
3315       \int_set:Nn \l_tmpa_int 1
3316     }
3317     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3318   }
3319   {
3320     \str_if_eq:eeTF t \l_@@_baseline_tl
3321     { \int_set:Nn \l_tmpa_int 1 }
3322     {
3323       \str_if_eq:eeTF b \l_@@_baseline_tl
3324       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3325       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3326     }
3327     \bool_lazy_or:nnT
3328       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3329       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3330     {
3331       \@@_error:n { bad-value-for-baseline }
3332       \int_set:Nn \l_tmpa_int 1
3333     }
3334     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3335     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3336   }
3337   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3338   \endpgfpicture
3339   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3340 }

```

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).

```

3341 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3342 {

```

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.

```

3343   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3344   {
3345     \int_compare:nNnT \c@jCol > 1
3346     {
3347       \box_set_wd:Nn \l_@@_the_array_box
3348       { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3349     }
3350   }

```

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

```

3351   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3352   \bool_if:NT \l_@@_caption_above_bool
3353   {
3354     \tl_if_empty:NF \l_@@_caption_tl
3355     {
3356       \bool_set_false:N \g_@@_caption_finished_bool
3357       \int_gzero:N \c@tabularnote
3358       \@@_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).

```

3359       \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3360       {
3361         \tl_gput_right:Ne \g_@@_aux_tl
3362         {
3363           \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3364           { \int_use:N \g_@@_notes_caption_int }
3365         }
3366         \int_gzero:N \g_@@_notes_caption_int
3367       }
3368     }
3369   }

```

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

```

3370   \hbox
3371   {
3372     \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.

```

3373     \@@_create_extra_nodes:
3374     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3375   }

```

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).

```

3376   \bool_lazy_any:nT
3377   {
3378     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3379     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3380     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3381   }
3382   {
3383     \bool_if:NTF \l_@@_notes_no_print_bool
3384     { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }

```

```

3385         \@@_tabular_notes:
3386     }
3387     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3388     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3389     \end { minipage }
3390 }

```

```

3391 \cs_new_protected:Npn \@@_insert_caption:
3392 {
3393     \tl_if_empty:NF \l_@@_caption_tl
3394     {
3395         \cs_if_exist:NTF \@capttype
3396         { \@@_insert_caption_i: }
3397         { \@@_error:n { caption~outside~float } }
3398     }
3399 }

```

```

3400 \cs_new_protected:Npn \@@_insert_caption_i:
3401 {
3402     \group_begin:

```

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

```

3403     \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.

```

3404     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3405     \tl_if_empty:NTF \l_@@_short_caption_tl
3406     \caption
3407     { \caption [ \l_@@_short_caption_tl ] }
3408     { \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.

```

3409     \bool_if:NF \g_@@_caption_finished_bool
3410     {
3411         \bool_gset_true:N \g_@@_caption_finished_bool
3412         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3413         \int_gzero:N \c@tabularnote
3414     }
3415     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3416     \group_end:
3417 }

```

```

3418 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3419 {
3420     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3421     \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3422 }

```

```

3423 \cs_set_protected:Npn \@@_tabular_notes_error:
3424 { \@@_error:n { Misuse~of~NiceTabularNotes } }

```

```

3425 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:

```

```

3426 \cs_set_protected:Npn \@@_tabular_notes:
3427 {
3428     \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3429     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3430     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3431     \skip_vertical:N 0.65ex

```

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

```

3432   \group_begin:
3433   \l_@@_notes_code_before_tl
3434   \tl_if_empty:NF \g_@@_tabularnote_tl
3435   {
3436     \g_@@_tabularnote_tl \par
3437     \tl_gclear:N \g_@@_tabularnote_tl
3438   }

```

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.

```

3439   \int_compare:nNnT \c@tabularnote > \c_zero_int
3440   {
3441     \bool_if:NTF \l_@@_notes_para_bool
3442     {
3443       \begin { tabularnotes* }
3444       \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3445       \strut
3446       \end { tabularnotes* }

```

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

```

3447       \par
3448     }
3449     {
3450       \tabularnotes
3451       \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3452       \strut
3453       \endtabularnotes
3454     }
3455   }
3456   \unskip
3457   \group_end:
3458   \bool_if:NT \l_@@_notes_bottomrule_bool
3459   {
3460     \IfPackageLoadedTF { booktabs }
3461     {

```

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

```

3462       \skip_vertical:N \aboverulesep

```

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

```

3463       { \CT@arc@ \hrule height \heavyrulewidth }
3464     }
3465     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3466   }
3467   \l_@@_notes_code_after_tl
3468   \seq_gclear:N \g_@@_notes_seq
3469   \seq_gclear:N \g_@@_notes_in_caption_seq
3470   \int_gzero:N \c@tabularnote
3471 }

```

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.

```

3472 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3473 {
3474   \tl_if_novalue:nTF { #1 }
3475   { \item }
3476   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3477 }

```

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`.

```

3478 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3479 {
3480   \pgfpicture
3481     \@@_qpoint:n { row - 1 }
3482     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3483     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3484     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3485   \endpgfpicture
3486   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3487   \int_if_zero:nT \l_@@_first_row_int
3488   {
3489     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3490     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3491   }
3492   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3493 }

```

Now, the general case.

```

3494 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3495 {

```

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

```

3496   \str_if_eq:eeT \l_@@_baseline_tl { t }
3497   { \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`.

```

3498   \pgfpicture
3499   \@@_qpoint:n { row - 1 }
3500   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3501   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3502   {
3503     \int_set:Nn \l_tmpa_int
3504     {
3505       \str_range:Nnn
3506         \l_@@_baseline_tl
3507         { 6 }
3508         { \tl_count:o \l_@@_baseline_tl }
3509     }
3510     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3511   }
3512   {
3513     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3514     \bool_lazy_or:nnT
3515       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3516       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3517     {
3518       \@@_error:n { bad-value-for-baseline }
3519       \int_set:Nn \l_tmpa_int 1
3520     }
3521     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3522   }
3523   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3524   \endpgfpicture
3525   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3526   \int_if_zero:nT \l_@@_first_row_int
3527   {
3528     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3529     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3530   }
3531   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3532 }
```

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.

```
3533 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3534 {
```

We will compute the real width of both delimiters used.

```
3535 \dim_zero_new:N \l_@@_real_left_delim_dim
3536 \dim_zero_new:N \l_@@_real_right_delim_dim
3537 \hbox_set:Nn \l_tmpb_box
3538 {
3539   \m@th
3540   $ % $
3541   \left #1
3542   \vcenter
3543   {
3544     \vbox_to_ht:nn
3545     { \box_ht_plus_dp:N \l_tmpa_box }
3546     { }
3547   }
3548   \right .
3549   $ % $
3550 }
3551 \dim_set:Nn \l_@@_real_left_delim_dim
3552 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3553 \hbox_set:Nn \l_tmpb_box
3554 {
3555   \m@th
3556   $ % $
3557   \left .
3558   \vbox_to_ht:nn
3559   { \box_ht_plus_dp:N \l_tmpa_box }
3560   { }
3561   \right #2
3562   $ % $
3563 }
3564 \dim_set:Nn \l_@@_real_right_delim_dim
3565 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3566 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3567 \@@_put_box_in_flow:
3568 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3569 }
```

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).

```
3570 \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).

```
3571 {
3572   \peek_remove_spaces:n
3573   {
3574     \peek_meaning:NTF \end
3575     \@@_analyze_end:Nn
```

```

3576         {
3577             \@@_transform_preamble:
3578             \@@_array:o \l_@@_array_preamble_tl
3579         }
3580     }
3581 }
3582 {
3583     \@@_create_col_nodes:
3584     \endarray
3585 }

```

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

```

3586 \NewDocumentEnvironment { @@-light-syntax } { b }
3587 {

```

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`.

```

3588     \tl_if_empty:nT { #1 }
3589     { \@@_fatal:n { empty-environment } }
3590     \tl_if_in:nnT { #1 } { & }
3591     { \@@_fatal:n { ampersand-in-light-syntax } }
3592     \tl_if_in:nnT { #1 } { \ }
3593     { \@@_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`.

```

3594     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3595 }

```

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.

```

3596 {
3597     \@@_create_col_nodes:
3598     \endarray
3599 }
3600 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3601 {
3602     \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).

```

3603     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3604     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3605     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3606     \seq_set_split:Nee
3607     \seq_set_split:Non
3608     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3609     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3610     \tl_if_empty:NF \l_tmpa_tl
3611     { \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.

```

3612   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3613   { \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`).

```

3614   \tl_build_begin:N \l_@@_new_body_tl
3615   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3616   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3617   \@@_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).

```

3618   \seq_map_inline:Nn \l_@@_rows_seq
3619   {
3620     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3621     \@@_line_with_light_syntax:n { ##1 }
3622   }
3623   \tl_build_end:N \l_@@_new_body_tl
3624   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3625   {
3626     \int_set:Nn \l_@@_last_col_int
3627     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3628   }

```

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.

```

3629   \@@_transform_preamble:

3630   \@@_array:o \l_@@_array_preamble_tl \l_@@_new_body_tl
3631   }

3632   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3633   {
3634     \seq_clear_new:N \l_@@_cells_seq
3635     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3636     \int_set:Nn \l_@@_nb_cols_int
3637     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3638     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3639     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3640     \seq_map_inline:Nn \l_@@_cells_seq
3641     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3642   }
3643   \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`.

```

3644   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3645   {
3646     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3647     { \@@_fatal:n { empty-environment } }

```

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

```

3648   \end { #2 }
3649   }

```

The following command will be used in `\@@_create_col_nodes`.

```

3650 \socket_new:nn { nicematrix / horizontal-space } { 0 }
3651 \socket_new_plug:nnn { nicematrix / horizontal-space } { standard }
3652 { \skip_horizontal:N \g_tmpa_skip }
3653 \socket_assign_plug:nn { nicematrix / horizontal-space } { standard }
3654 \socket_new_plug:nnn { nicematrix / horizontal-space } { with-false-col }
3655 {
3656   \clist_if_in:N\TF \g_@@_false_cols_clist { \int_use:N \g_tmpa_int }
3657   {
3658     % modified 2026/09/03
3659     \skip_set:Nn \l_tmpa_skip { 0 pt-plus 1 fill }
3660     % \skip_set:Nn \l_tmpa_skip { 0 pt }
3661     \skip_add:Nn \l_tmpa_skip \l_@@_width_of_false_dim
3662     \skip_add:Nn \l_tmpa_skip \arrayrulewidth
3663     \skip_horizontal:N \l_tmpa_skip
3664   }
3665   { \skip_horizontal:N \g_tmpa_skip }
3666 }

```

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`).

```

3667 \cs_new:Npn \@@_create_col_nodes:
3668 {
3669   \crrc
3670   \int_if_zero:nT \l_@@_first_col_int
3671   {
3672     \omit
3673     \hbox_overlap_left:n
3674     {
3675       \bool_if:NT \l_@@_code_before_bool
3676       { \pgfsys@markposition { \@@_env: - col - 0 } }
3677       \pgfpicture
3678       \pgfrememberpicturepositiononpagetrue
3679       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3680       \str_if_empty:NF \l_@@_name_str
3681       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3682       \endpgfpicture
3683       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3684     }
3685     &
3686   }
3687   \omit

```

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

```

3688   \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`).

```

3689   \int_if_zero:nTF \l_@@_first_col_int
3690   {
3691     \@@_mark_position:n { 1 }
3692     \pgfpicture
3693     \pgfrememberpicturepositiononpagetrue
3694     \pgfcoordinate { \@@_env: - col - 1 }
3695     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3696     \str_if_empty:NF \l_@@_name_str
3697     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3698     \endpgfpicture
3699   }
3700   {
3701     \bool_if:NT \l_@@_code_before_bool
3702     {

```

```

3703         \hbox
3704         {
3705             \skip_horizontal:n { 0.5 \arrayrulewidth }
3706             \pgfsys@markposition { \@@_env: - col - 1 }
3707             \skip_horizontal:n { -0.5 \arrayrulewidth }
3708         }
3709     }
3710     \pgfpicture
3711     \pgfrememberpicturepositiononpagetrue
3712     \pgfcoordinate { \@@_env: - col - 1 }
3713     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3714     \@@_node_alias:n { 1 }
3715     \endpgfpicture
3716 }

```

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.

```

3717 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3718 \bool_if:NTF \l_@@_row_columns_width_bool
3719 { \skip_gadd:Nn \g_tmpa_skip { \box_ht_plus_dp:N \@arstrutbox } }
3720 {
3721     \bool_lazy_or:nnT
3722     { \l_@@_auto_columns_width_bool }
3723     { \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim }
3724     {
3725         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3726         \bool_lazy_and:nnTF
3727         \l_@@_auto_columns_width_bool
3728         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3729         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3730         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3731     }
3732 }
3733 \int_gset:Nn \g_tmpa_int 1
3734 \socket_use:n { nicematrix / horizontal-space }
3735 \hbox
3736 {
3737     \@@_mark_position:n { 2 }
3738     \pgfpicture
3739     \pgfrememberpicturepositiononpagetrue
3740     \pgfcoordinate { \@@_env: - col - 2 }
3741     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3742     \@@_node_alias:n { 2 }
3743     \endpgfpicture
3744 }

```

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.

```

3745 \bool_if:NTF \g_@@_last_col_found_bool
3746 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3747 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3748 {
3749     &
3750     \omit
3751     \int_gincr:N \g_tmpa_int

```

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

```

3752     \socket_use:n { nicematrix / horizontal-space }
3753     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3754 \pgfpicture
3755 \pgfrememberpicturepositiononpagetrue
3756 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3757 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3758 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3759 \endpgfpicture
3760 }

```

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).

```

3761 \bool_lazy_or:nnF
3762 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3763 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3764 {
3765   &
3766   \omit
3767   \skip_horizontal:N \g_tmpa_skip
3768   \int_gincr:N \g_tmpa_int
3769   \bool_lazy_any:nF
3770   {
3771     \g_@@_delims_bool
3772     \l_@@_tabular_bool
3773     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3774     \l_@@_exterior_arraycolsep_bool
3775     \l_@@_bar_at_end_of_pream_bool
3776   }
3777   { \skip_horizontal:n { - \col@sep } }
3778   \bool_if:NT \l_@@_code_before_bool
3779   {
3780     \hbox
3781     {
3782       \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.

```

3783 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3784 { \skip_horizontal:n { - \arraycolsep } }
3785 \pgfsys@markposition
3786 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3787 \skip_horizontal:n { 0.5 \arrayrulewidth }
3788 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3789 { \skip_horizontal:N \arraycolsep }
3790 }
3791 }
3792 \pgfpicture
3793 \pgfrememberpicturepositiononpagetrue
3794 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3795 {
3796   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3797   {
3798     \pgfpoint
3799     { - 0.5 \arrayrulewidth - \arraycolsep }
3800     \c_zero_dim
3801   }
3802   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3803 }
3804 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3805 \endpgfpicture
3806 }

```

```

3807 \bool_if:NT \g_@@_last_col_found_bool

```

```

3808 {
3809   \hbox_overlap_right:n
3810   {
3811     \skip_horizontal:N \g_@@_width_last_col_dim
3812     \skip_horizontal:N \col@sep
3813     \bool_if:NT \l_@@_code_before_bool
3814     {
3815       \pgfsys@markposition
3816       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3817     }
3818     \pgfpicture
3819     \pgfrememberpicturepositiononpagetrue
3820     \pgfcoordinate
3821     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3822     \pgfpointorigin
3823     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3824     \endpgfpicture
3825   }
3826 }
3827 }

3828 \cs_new_protected:Npn \@@_mark_position:n #1
3829 {
3830   \bool_if:NT \l_@@_code_before_bool
3831   {
3832     \hbox
3833     {
3834       \skip_horizontal:n { -0.5 \arrayrulewidth }
3835       \pgfsys@markposition { \@@_env: - col - #1 }
3836       \skip_horizontal:n { 0.5 \arrayrulewidth }
3837     }
3838   }
3839 }

3840 \cs_new_protected:Npn \@@_node_alias:n #1
3841 {
3842   \str_if_empty:NF \l_@@_name_str
3843   { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3844 }

```

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

```

3845 \tl_const:Nn \c_@@_preamble_first_col_tl
3846 {
3847   >
3848   {

```

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

```

3849   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3850   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3851   \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:
3852   \bool_gset_true:N \g_@@_after_col_zero_bool
3853   \@@_begin_of_row:
3854   \hbox_set:Nw \l_@@_cell_box
3855   \@@_math_toggle:
3856   \@@_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”.

```

3857   \int_compare:nNnT \c@iRow > \c_zero_int
3858   {
3859     \bool_lazy_or:nnT

```

```

3860         { \int_compare:nNn \l_@@_last_row_int < \c_zero_int }
3861         { \int_compare:nNn \c_iRow < \l_@@_last_row_int }
3862         {
3863             \l_@@_code_for_first_col_tl
3864             \xglobal \colorlet { nicematrix-first-col } { . }
3865         }
3866     }
3867 }

```

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`.

```

3868     1
3869     <
3870     {
3871         \@@_math_toggle:
3872         \hbox_set_end:
3873         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3874         \@@_adjust_size_box:
3875         \@@_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.

```

3876         \dim_gset:Nn \g_@@_width_first_col_dim
3877         { \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.

```

3878         \hbox_overlap_left:n
3879         {
3880             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3881             \@@_node_cell:
3882             { \box_use_drop:N \l_@@_cell_box }
3883             \skip_horizontal:N \l_@@_left_delim_dim
3884             \skip_horizontal:N \l_@@_left_margin_dim
3885             \skip_horizontal:N \l_@@_extra_left_margin_dim
3886         }
3887         \bool_gset_false:N \g_@@_empty_cell_bool
3888         \skip_horizontal:n { -2 \col@sep }
3889     }
3890 }

```

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

```

3891 \tl_const:Nn \c_@@_preamble_last_col_tl
3892 {
3893     >
3894     {
3895         \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).

```

3896         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3897         \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3898         \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

3899         \bool_gset_true:N \g_@@_last_col_found_bool
3900         \int_gincr:N \c@jCol
3901         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3902         \hbox_set:Nw \l_@@_cell_box
3903         \@@_math_toggle:
3904         \@@_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”.

```

3905         \int_compare:nNnT \c_iRow > \c_zero_int
3906         {
3907             \bool_lazy_or:nnT

```

```

3908         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3909         { \int_compare_p:nNn \c_iRow < \l_@@_last_row_int }
3910         {
3911             \l_@@_code_for_last_col_tl
3912             \xglobal \colorlet { nicematrix-last-col } { . }
3913         }
3914     }
3915 }
3916 1
3917 <
3918 {
3919     \@@_math_toggle:
3920     \hbox_set_end:
3921     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3922     \@@_adjust_size_box:
3923     \@@_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.

```

3924     \dim_gset:Nn \g_@@_width_last_col_dim
3925     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3926     \skip_horizontal:n { -2 \col@sep }

```

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

```

3927     \hbox_overlap_right:n
3928     {
3929         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3930         {
3931             \skip_horizontal:N \l_@@_right_delim_dim
3932             \skip_horizontal:N \l_@@_right_margin_dim
3933             \skip_horizontal:N \l_@@_extra_right_margin_dim
3934             \@@_node_cell:
3935         }
3936     }
3937     \bool_gset_false:N \g_@@_empty_cell_bool
3938 }
3939 }

```

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

```

3940 \NewDocumentEnvironment { NiceArray } { }
3941 {
3942     \bool_gset_false:N \g_@@_delims_bool
3943     \str_if_empty:NT \g_@@_name_env_str
3944     { \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).

```

3945     \NiceArrayWithDelims . .
3946 }
3947 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3948 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3949 {
3950     \NewDocumentEnvironment { #1 NiceArray } { }
3951     {
3952         \bool_gset_true:N \g_@@_delims_bool
3953         \str_if_empty:NT \g_@@_name_env_str
3954         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3955         \@@_test_if_math_mode:
3956         \NiceArrayWithDelims #2 #3
3957     }

```

```

3958     { \endNiceArrayWithDelims }
3959 }

3960 \@@_def_env:NNN p (      )
3961 \@@_def_env:NNN b [      ]
3962 \@@_def_env:NNN B \{      \}
3963 \@@_def_env:NNN v \vert \vert
3964 \@@_def_env:NNN V \Vert \Vert

```

13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3965 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3966 {
3967   \bool_set_false:N \l_@@_preamble_bool
3968   \tl_clear:N \l_tmpa_tl
3969   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3970     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3971   \tl_put_right:Nn \l_tmpa_tl
3972   {
3973     *
3974     {
3975       \int_case:nnF \l_@@_last_col_int
3976       {
3977         { -2 } { \c@MaxMatrixCols }
3978         { -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).

```

3979       }
3980       { \int_eval:n { \l_@@_last_col_int - 1 } }
3981     }
3982     { #2 }
3983   }
3984   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3985   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3986 }
3987 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3988 \clist_map_inline:nn { p , b , B , v , V }
3989 {
3990   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3991   {
3992     \bool_gset_true:N \g_@@_delims_bool
3993     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3994     \int_if_zero:nT \l_@@_last_col_int
3995     {
3996       \bool_set_true:N \l_@@_last_col_without_value_bool
3997       \int_set:Nn \l_@@_last_col_int { -1 }
3998     }
3999     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
4000     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
4001   }
4002   { \use:c { end #1 NiceArray } }
4003 }

```

We define also an environment {NiceMatrix}

```

4004 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
4005 {
4006   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }

```

```

4007 \int_if_zero:nT \l_@@_last_col_int
4008 {
4009     \bool_set_true:N \l_@@_last_col_without_value_bool
4010     \int_set:Nn \l_@@_last_col_int { -1 }
4011 }
4012 \keys_set:nn { nicematrix / NiceMatrix } { #1 }
4013 \bool_lazy_or:nnT
4014     \l_@@_except_borders_bool
4015     { \clist_if_empty_p:N \l_@@_vlines_clist }
4016     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
4017 \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
4018 }
4019 { \endNiceArray }

```

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

```

4020 \cs_new_protected:Npn \@@_NotEmpty:
4021 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

13.2 The key renew-matrix

```

4022 \cs_set_protected:Npn \@@_renew_matrix:
4023 {
4024     \tl_map_inline:nn { pvVbB }
4025     { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
4026 }

```

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

```

4027 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
4028 {

```

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`.

```

4029 \dim_compare:nNnT\l_@@_width_dim = \c_zero_dim
4030 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
4031 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
4032 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
4033 \tl_if_empty:NF \l_@@_short_caption_tl
4034 {
4035     \tl_if_empty:NT \l_@@_caption_tl
4036     {
4037         \@@_error_or_warning:n { short-caption-without~caption }
4038         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
4039     }
4040 }
4041 \tl_if_empty:NF \l_@@_label_tl
4042 {
4043     \tl_if_empty:NT \l_@@_caption_tl
4044     { \@@_error_or_warning:n { label~without~caption } }
4045 }
4046 \NewDocumentEnvironment { TabularNote } { b }
4047 {
4048     \bool_if:NTF \l_@@_in_code_after_bool
4049     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
4050     {
4051         \tl_if_empty:NF \g_@@_tabularnote_tl
4052         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
4053         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
4054     }
4055 }
4056 { }

```

```

4057 \@@_settings_for_tabular:
4058 \NiceArray { #2 }
4059 }
4060 { \endNiceArray }
4061 \cs_new_protected:Npn \@@_settings_for_tabular:
4062 {
4063 \bool_set_true:N \l_@@_tabular_bool
4064 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
4065 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
4066 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
4067 }

4068 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
4069 {
4070 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
4071 \dim_set:Nn \l_@@_width_dim { #1 }
4072 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4073 \@@_settings_for_tabular:
4074 \NiceArray { #3 }
4075 }
4076 {
4077 \endNiceArray
4078 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
4079 { \@@_error_or_warning:n { NiceTabularX~without~X } }
4080 }

4081 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
4082 {
4083 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
4084 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
4085 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4086 \@@_settings_for_tabular:
4087 \NiceArray { #3 }
4088 }
4089 { \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`).

```

4090 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
4091 {
4092 \bool_lazy_all:nT
4093 {
4094 { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
4095 { \l_@@_hvlines_bool }
4096 { ! \g_@@_delims_bool }
4097 { ! \l_@@_except_borders_bool }
4098 }
4099 {
4100 \bool_set_true:N \l_@@_except_borders_bool
4101 \clist_if_empty:NF \l_@@_corners_clist
4102 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
4103 \tl_gput_right:Nn \g_@@_rules_tl
4104 {
4105 \@@_stroke_block:nnnnn
4106 {
4107 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,

```

```

4108         draw = \l_@@_rules_color_tl
4109     }
4110     { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4111 }
4112 }
4113 }

```

```

4114 \cs_new_protected:Npn \@@_after_array:
4115 {
4116     \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.

```

4117     \bool_if:NT \g_@@_last_col_found_bool
4118     { \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`.

```

4119     \bool_if:NT \l_@@_last_col_without_value_bool
4120     { \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.

```

4121     \bool_if:NT \l_@@_last_row_without_value_bool
4122     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

4123     \tl_gput_right:Ne \g_@@_aux_tl
4124     {
4125         \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4126         {
4127             \int_use:N \l_@@_first_row_int ,
4128             \int_use:N \c@iRow ,
4129             \int_use:N \g_@@_row_total_int ,
4130             \int_use:N \l_@@_first_col_int ,
4131             \int_use:N \c@jCol ,
4132             \int_use:N \g_@@_col_total_int
4133         }
4134     }

4135     \clist_if_empty:NF \g_@@_false_cols_clist
4136     {
4137         \tl_gput_right:Ne \g_@@_aux_tl
4138         {
4139             \clist_set:Nn \exp_not:N \g_@@_false_cols_clist
4140             { \g_@@_false_cols_clist }
4141         }
4142     }
4143     \clist_if_empty:NF \g_@@_false_rows_clist
4144     {
4145         \tl_gput_right:Ne \g_@@_aux_tl
4146         {
4147             \clist_set:Nn \exp_not:N \g_@@_false_rows_clist
4148             { \g_@@_false_rows_clist }
4149         }
4150     }

4151     \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`).

```

4152   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4153   {
4154     \tl_gput_right:Ne \g_@@_aux_tl
4155     {
4156       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4157       { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4158     }
4159   }
4160   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4161   {
4162     \tl_gput_right:Ne \g_@@_aux_tl
4163     {
4164       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4165       { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4166       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4167       { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4168     }
4169   }

```

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

```

4170   \@@_create_diag_nodes:

```

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

```

4171   \pgfpicture
4172   \@@_create_aliases_last:
4173   \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4174   \endpgfpicture

4175   \bool_lazy_and:nnF
4176   { \clist_if_empty_p:N \g_@@_false_cols_clist }
4177   { \clist_if_empty_p:N \g_@@_false_rows_clist }
4178   {
4179     \@@_mark_blocks:
4180     \@@_treat_false_cols:
4181     \@@_treat_false_rows:
4182   }

```

By default, the diagonal lines will be parallelized¹³. 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.

```

4183   \bool_if:NT \l_@@_parallelize_diags_bool
4184   {
4185     \int_gzero:N \g_@@_ddots_int
4186     \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.

```

4187     \dim_gzero:N \g_@@_delta_x_one_dim
4188     \dim_gzero:N \g_@@_delta_y_one_dim
4189     \dim_gzero:N \g_@@_delta_x_two_dim
4190     \dim_gzero:N \g_@@_delta_y_two_dim
4191   }

4192   \bool_set_false:N \l_@@_initial_open_bool
4193   \bool_set_false:N \l_@@_final_open_bool

```

¹³It's possible to use the option `parallelize-diags` to disable this parallelization.

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.

```
4194 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
4195 \@@_draw_dotted_lines:
```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` but (for efficiency) they will also be marked directly in the main hash table of TeX.

```
4196 \clist_if_empty:NF \l_@@_corners_clist
4197 {
4198   \bool_if:NTF \l_@@_no_cell_nodes_bool
4199   { \@@_error:n { corners~with~no~cell~nodes } }
4200   \@@_compute_corners:
4201 }
```

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`.

```
4202 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4203   \g_@@_pos_of_blocks_seq
4204   \g_@@_future_pos_of_blocks_seq
4205 \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-*}`).

```
4206 \@@_adjust_pos_of_blocks_seq:
4207 \@@_deal_with_rounded_corners:
4208 \legacy_if:nF { measuring@ } \@@_draw_rules:
4209 \tl_gclear:N \g_@@_rules_tl
```

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

```
4210 \IfPackageLoadedT { tikz }
4211 {
4212   \tikzset
4213   {
4214     every-picture / .style =
4215     {
4216       overlay ,
4217       remember-picture ,
4218       name-prefix = \@@_env: -
4219     }
4220   }
4221 }
4222 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4223 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4224 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4225 \cs_set_eq:NN \OverBrace \@@_OverBrace
4226 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4227 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4228 \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.

```
4229 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4230 \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.

```
4231 \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).

```
4232 \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`).

```
4233 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4234 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }
4235 \bool_set_true:N \l_@@_in_code_after_bool
4236 \@@_security_font_commands:
```

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:`.

```
4237 \bool_set_true:N \l_@@_in_code_after_bool
4238 \if_mode_math:
4239 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4240 \scan_stop:
4241 \else:
4242 $ % $
4243 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4244 \scan_stop:
4245 $ % $
4246 \fi:
4247 \tl_gclear:N \g_nicematrix_code_after_tl
4248 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4249 \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.

```
4250 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4251 \tl_if_empty:NF \g_@@_pre_code_before_tl
4252 {
4253 \tl_gput_right:Ne \g_@@_aux_tl
4254 {
4255 \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4256 { \exp_not:o \g_@@_pre_code_before_tl }
4257 }
4258 \tl_gclear:N \g_@@_pre_code_before_tl
4259 }
4260 \tl_if_empty:NF \g_nicematrix_code_before_tl
4261 {
4262 \tl_gput_right:Ne \g_@@_aux_tl
4263 {
4264 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4265 { \exp_not:o \g_nicematrix_code_before_tl }
4266 }
4267 \tl_gclear:N \g_nicematrix_code_before_tl
4268 }

4269 \str_gclear:N \g_@@_name_env_str
4270 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹⁴. 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 use the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

4271   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4272 }

4273 \cs_new_protected:Npn \@@_test_false_columns:
4274 {
4275   \clist_map_inline:Nn \g_@@_false_cols_clist
4276   {
4277     \int_step_inline:nnn { \l_@@_first_row_int } { \arabic { iRow } }
4278     {
4279       \cs_if_exist:cT
4280       { pgf @ sh @ ns @ \@@_env: - ####1 - ##1 }
4281       {
4282         \@@_error:nnn { Not~empty~cell~in~false~column }
4283         { ####1 }
4284         { ##1 }
4285       }
4286     }
4287   }
4288 }

4289 \cs_new_protected:Npn \@@_treat_false_rows:
4290 { \clist_map_function:NN \g_@@_false_rows_clist \@@_treat_one_false_row:n }

4291 \cs_new_protected:Npn \@@_treat_one_false_row:n #1
4292 {
4293   \int_zero:N \l_tmpa_int
4294   \int_step_inline:nn \c@jCol
4295   {
4296     \int_if_zero:nTF \l_tmpa_int
4297     {
4298       \@@_if_in_block:nnF { #1 } { ##1 }
4299       { \int_set:Nn \l_tmpa_int { ##1 } }
4300     }
4301     {
4302       \@@_if_in_block:nnT { #1 } { ##1 }
4303       {
4304         \@@_false_h_segment:nee
4305         { #1 }
4306         { \int_use:N \l_tmpa_int }
4307         { \int_eval:n { ##1 - 1 } }
4308         \int_zero:N \l_tmpa_int
4309       }
4310     }
4311   }
4312   \int_compare:nNnT \l_tmpa_int > 0
4313   {
4314     \clist_if_in:NneTF \g_@@_false_cols_clist { \int_use:N \c@jCol }
4315     { \int_set_eq:NN \l_tmpb_int \c@jCol }
4316     { \int_set:Nn \l_tmpb_int { \c@jCol + 1 } }
4317     \@@_false_h_segment:nee { #1 }
4318     { \int_use:N \l_tmpa_int }
4319     { \int_use:N \l_tmpb_int }
4320   }
4321 }

```

¹⁴e.g. `\color[rgb]{0.5,0.5,0}`

```

4322 \cs_new_protected:Npn \@@_false_h_segment:nnn #1 #2 #3
4323 {

```

The command `\clist_if_in:NnTF` is *not* expandable and that's why we have to transit by a variable `\l_tmpa_int`.

```

4324   \int_compare:nNnTF { #2 } = 1
4325   {
4326     \clist_if_in:NnTF \g_@@_false_cols_clist { 1 }
4327     { \int_set:Nn \l_tmpa_int 1 }
4328     { \int_zero:N \l_tmpa_int }
4329   }
4330   { \int_set:Nn \l_tmpa_int { #2 } }
4331   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4332   {
4333     { #1 }
4334     { \int_use:N \l_tmpa_int }
4335     { #1 }
4336     { #3 }
4337     { }
4338   }
4339   \tl_gput_right:Ne \g_@@_pre_code_before_tl
4340   { \@@_rectanglecolor { \l_@@_color_of_false_tl } { #1 - #2 } { #1 - #3 } }
4341 }
4342 \cs_generate_variant:Nn \@@_false_h_segment:nnn { n e e }
4343 \cs_new_protected:Npn \@@_treat_false_cols:
4344 { \clist_map_function:NN \g_@@_false_cols_clist \@@_treat_one_false_col:n }
4345 \cs_new_protected:Npn \@@_treat_one_false_col:n #1
4346 {
4347   \int_zero:N \l_tmpa_int
4348   \int_step_inline:nn \c@iRow
4349   {
4350     \int_if_zero:nTF \l_tmpa_int
4351     {
4352       \@@_if_in_block:nnF { ##1 } { #1 }
4353       { \int_set:Nn \l_tmpa_int { ##1 } }
4354     }
4355     {
4356       \@@_if_in_block:nnT { ##1 } { #1 }
4357       {
4358         \@@_false_v_segment:nee
4359         { #1 }
4360         { \int_use:N \l_tmpa_int }
4361         { \int_eval:n { ##1 - 1 } }
4362         \int_zero:N \l_tmpa_int
4363       }
4364     }
4365   }
4366 }
4367 \int_compare:nNnT \l_tmpa_int > 0
4368 {
4369   \clist_if_in:NnTF \g_@@_false_rows_clist { \int_use:N \c@iRow }
4370   { \int_set_eq:NN \l_tmpb_int \c@iRow }
4371   { \int_set:Nn \l_tmpb_int { \c@iRow + 1 } }
4372   \@@_false_v_segment:nee { #1 }
4373   { \int_use:N \l_tmpa_int }
4374   { \int_use:N \l_tmpb_int }
4375 }
4376 }
4377 \cs_new_protected:Npn \@@_false_v_segment:nnn #1 #2 #3
4378 {
4379   \int_compare:nNnTF { #2 } = 1

```

```

4380 {
4381   \clist_if_in:NnTF \g_@@_false_rows_clist { 1 }
4382   { \int_set:Nn \l_tmpa_int 1 }
4383   { \int_zero:N \l_tmpa_int }
4384 }
4385 { \int_set:Nn \l_tmpa_int { #2 } }
4386 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4387 {
4388   { \int_use:N \l_tmpa_int }
4389   { #1 }
4390   { #3 }
4391   { #1 }
4392   { }
4393 }
4394 \tl_gput_right:Ne \g_@@_pre_code_before_tl
4395 { \@@_rectanglecolor { \l_@@_color_of_false_tl } { #2 - #1 } { #3 - #1 } }
4396 }
4397 \cs_generate_variant:Nn \@@_false_v_segment:nnn { n e e }

4398 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4399 {
4400   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4401   \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.

```

4402   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4403   { 0.6 \l_@@_xdots_shorten_start_dim }
4404   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4405   { 0.6 \l_@@_xdots_shorten_end_dim }
4406 }

```

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`.

```

4407 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4408 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4409 \cs_new_protected:Npn \@@_create_alias_nodes:
4410 {
4411   \int_step_inline:nn \c@iRow
4412   {
4413     \pgfnodealias
4414     { \l_@@_name_str - ##1 - last }
4415     { \@@_env: - ##1 - \int_use:N \c@jCol }
4416   }
4417   \int_step_inline:nn \c@jCol
4418   {
4419     \pgfnodealias
4420     { \l_@@_name_str - last - ##1 }
4421     { \@@_env: - \int_use:N \c@iRow - ##1 }
4422   }
4423   \pgfnodealias
4424   { \l_@@_name_str - last - last }
4425   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4426 }

```

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).

```

4427 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4428 {
4429   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4430   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4431 }

```

The following command must *not* be protected.

```

4432 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4433 {
4434   { #1 }
4435   { #2 }
4436   {
4437     \int_compare:nNnTF { #3 } > { 98 }
4438     { \int_use:N \c@iRow }
4439     { #3 }
4440   }
4441   {
4442     \int_compare:nNnTF { #4 } > { 98 }
4443     { \int_use:N \c@jCol }
4444     { #4 }
4445   }
4446   { #5 }
4447 }

```

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).

```

4448 \AtBeginDocument
4449 {
4450   \cs_new_protected:Npe \@@_draw_dotted_lines:
4451   {
4452     \c_@@_pgfortikzpicture_tl
4453     \@@_draw_dotted_lines_i:
4454     \c_@@_endpgfortikzpicture_tl
4455   }
4456 }

```

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

```

4457 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4458 {
4459   \pgfrememberpicturerepositiononpagetrue
4460   \pgf@relevantforpicturesizefalse
4461   \g_@@_HVdotsfor_lines_tl
4462   \g_@@_Vdots_lines_tl
4463   \g_@@_Ddots_lines_tl
4464   \g_@@_Iddots_lines_tl
4465   \g_@@_Cdots_lines_tl
4466   \g_@@_Ldots_lines_tl
4467 }

4468 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4469 {
4470   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4471   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4472 }

```

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

```

4473 \pgfdeclareshape { @@_diag_node }
4474 {
4475   \savedanchor { \five }
4476   {
4477     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4478     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4479   }
4480   \anchor { 5 } { \five }
4481   \anchor { center } { \pgfpointorigin }
4482   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4483   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4484   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4485   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4486   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4487   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4488   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4489   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4490   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4491   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4492 }

```

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).

```

4493 \cs_new_protected:Npn \@_create_diag_nodes:
4494 {
4495   \pgfpicture
4496   \pgfrememberpicturepositiononpagetrue
4497   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4498   {
4499     \@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4500     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4501     \@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4502     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4503     \@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4504     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4505     \@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4506     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4507     \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.

```

4508     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4509     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4510     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4511     \str_if_empty:NF \l_@@_name_str
4512     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4513   }

```

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

```

4514     \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4515     \@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4516     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4517     \@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4518     \pgfcoordinate
4519     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4520     \pgfnodealias
4521     { \@@_env: - last }
4522     { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4523     \str_if_empty:NF \l_@@_name_str
4524     {

```

```

4525     \pgfnodealias
4526     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4527     { \@@_env: - \int_use:N \l_tmpa_int }
4528     \pgfnodealias
4529     { \l_@@_name_str - last }
4530     { \@@_env: - last }
4531 }
4532 \endpgfpicture
4533 }

```

16 We draw the dotted lines

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

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

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

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

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

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

```

\cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
{
  \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
  \int_set:Nn \l_@@_initial_i_int { #1 }
  \int_set:Nn \l_@@_initial_j_int { #2 }
  \int_set:Nn \l_@@_final_i_int { #1 }
  \int_set:Nn \l_@@_final_j_int { #2 }
  \bool_set_false:N \l_@@_stop_loop_bool
  \bool_do_until:Nn \l_@@_stop_loop_bool
  {
    \int_add:Nn \l_@@_final_i_int { #3 }
    \int_add:Nn \l_@@_final_j_int { #4 }
    \bool_set_false:N \l_@@_final_open_bool
    \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
    {
      \int_compare:nNnTF { #3 } = { 1 }
      { \bool_set_true:N \l_@@_final_open_bool }
      {

```

```

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

```

```

\bool_set_false:N \l_@@_initial_open_bool
\int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
{
  \int_compare:nNnTF { #3 } = { 1 }
  { \bool_set_true:N \l_@@_initial_open_bool }
  {
    \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
    { \bool_set_true:N \l_@@_initial_open_bool }
  }
}
{
  \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
  {
    \int_compare: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.

```

4534 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4535 {

```

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

```

4536 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4537 \l_@@_initial_i_int = #1
4538 \l_@@_initial_j_int = #2
4539 \l_@@_final_i_int = #1
4540 \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.

```

4541 \let \l_@@_stop_loop_bool \c_false_bool
4542 \bool_do_until:Nn \l_@@_stop_loop_bool
4543 {

```

We test if we are still in the matrix.

```

4544 \advance \l_@@_final_i_int by #3
4545 \advance \l_@@_final_j_int by #4
4546 \let \l_@@_final_open_bool \c_false_bool
4547 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4548 \if_int_compare:w #3 = \c_one_int
4549 \let \l_@@_final_open_bool \c_true_bool
4550 \else:
4551 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4552 \let \l_@@_final_open_bool \c_true_bool
4553 \fi:
4554 \fi:
4555 \else:
4556 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4557 \if_int_compare:w #4 = -1
4558 \let \l_@@_final_open_bool \c_true_bool
4559 \fi:
4560 \else:
4561 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4562 \if_int_compare:w #4 = \c_one_int
4563 \let \l_@@_final_open_bool \c_true_bool
4564 \fi:
4565 \fi:
4566 \fi:
4567 \fi:
4568 \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.

```

4569 {

```

We do a step backwards.

```

4570 \advance \l_@@_final_i_int by - #3
4571 \advance \l_@@_final_j_int by - #4
4572 \let \l_@@_stop_loop_bool \c_true_bool
4573 }

```

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`.

```

4574     {
4575         \cs_if_exist:cTF
4576         {
4577             @@ _ dotted _
4578             \int_use:N \l_@@_final_i_int -
4579             \int_use:N \l_@@_final_j_int
4580         }
4581         {
4582             \advance \l_@@_final_i_int by - #3
4583             \advance \l_@@_final_j_int by - #4
4584             \let \l_@@_final_open_bool \c_true_bool
4585             \let \l_@@_stop_loop_bool \c_true_bool
4586         }
4587     {
4588         \cs_if_exist:cTF
4589         {
4590             pgf @ sh @ ns @ \@@_env:
4591             - \int_use:N \l_@@_final_i_int
4592             - \int_use:N \l_@@_final_j_int
4593         }
4594         { \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.

```

4595     {
4596         \cs_set_nopar:cpn
4597         {
4598             @@ _ dotted _
4599             \int_use:N \l_@@_final_i_int -
4600             \int_use:N \l_@@_final_j_int
4601         }
4602         { }
4603     }
4604 }
4605 }
4606 }

```

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

```

4607     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4608     \l_tmpa_int = \l_@@_col_min_int
4609     \advance \l_tmpa_int by -1
4610     \bool_do_until:Nn \l_@@_stop_loop_bool
4611     {
4612         \advance \l_@@_initial_i_int by - #3
4613         \advance \l_@@_initial_j_int by - #4
4614         \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.

```

4615     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4616     \if_int_compare:w #3 = \c_one_int
4617         \let \l_@@_initial_open_bool \c_true_bool
4618     \else:

```

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

```

4619         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4620         \let \l_@@_initial_open_bool \c_true_bool
4621     \fi:
4622 \fi:
4623 \else:
4624     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4625     \if_int_compare:w #4 = \c_one_int
4626     \let \l_@@_initial_open_bool \c_true_bool
4627     \fi:
4628 \else:
4629     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4630     \if_int_compare:w #4 = -1
4631     \let \l_@@_initial_open_bool \c_true_bool
4632     \fi:
4633 \fi:
4634 \fi:
4635 \fi:
4636 \bool_if:NTF \l_@@_initial_open_bool
4637 {
4638     \advance \l_@@_initial_i_int by #3
4639     \advance \l_@@_initial_j_int by #4
4640     \let \l_@@_stop_loop_bool \c_true_bool
4641 }
4642 {
4643     \cs_if_exist:cTF
4644     {
4645         @@ _ dotted _
4646         \int_use:N \l_@@_initial_i_int -
4647         \int_use:N \l_@@_initial_j_int
4648     }
4649     {
4650         \advance \l_@@_initial_i_int by #3
4651         \advance \l_@@_initial_j_int by #4
4652         \let \l_@@_initial_open_bool \c_true_bool
4653         \let \l_@@_stop_loop_bool \c_true_bool
4654     }
4655     {
4656         \cs_if_exist:cTF
4657         {
4658             pgf @ sh @ ns @ \@@_env:
4659             - \int_use:N \l_@@_initial_i_int
4660             - \int_use:N \l_@@_initial_j_int
4661         }
4662         { \let \l_@@_stop_loop_bool \c_true_bool }
4663         {
4664             \cs_set_nopar:cpn
4665             {
4666                 @@ _ dotted _
4667                 \int_use:N \l_@@_initial_i_int -
4668                 \int_use:N \l_@@_initial_j_int
4669             }
4670             { }
4671         }
4672     }
4673 }
4674 }
```

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.

```

4675 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4676 {
4677     { \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`.

```

4678     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4679     { \int_use:N \l_@@_final_i_int }
4680     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4681     { }
4682   }
4683 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analysis of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4684 \cs_new_protected:Npn \@@_open_shorten:
4685 {
4686   \bool_if:NT \l_@@_initial_open_bool
4687   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4688   \bool_if:NT \l_@@_final_open_bool
4689   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4690 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it's only the whole array (excepted exterior rows and columns).

```

4691 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4692 {
4693   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4694   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4695   \int_set_eq:NN \l_@@_row_max_int \c_iRow
4696   \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.

```

4697   \seq_if_empty:NTF \g_@@_submatrix_seq
4698   { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4699   {

```

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`.

```

4700     \seq_map_inline:Nn \g_@@_submatrix_seq
4701     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4702   }
4703 }

```

#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 programmation 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.

```

4704 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4705 {
4706   \if_int_compare:w #3 > #1
4707   \else:
4708     \if_int_compare:w #1 > #5
4709     \else:
4710       \if_int_compare:w #4 > #2
4711       \else:
4712         \if_int_compare:w #2 > #6
4713         \else:
4714           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4715           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4716           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4717           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4718         \fi:
4719       \fi:
4720     \fi:
4721   \fi:
4722 }

4723 \cs_new_protected:Npn \@@_set_initial_coords:
4724 {
4725   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4726   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4727 }
4728 \cs_new_protected:Npn \@@_set_final_coords:
4729 {
4730   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4731   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4732 }
4733 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4734 {
4735   \pgfpointanchor
4736   {
4737     \@@_env:
4738     - \int_use:N \l_@@_initial_i_int
4739     - \int_use:N \l_@@_initial_j_int
4740   }
4741   { #1 }
4742   \@@_set_initial_coords:
4743 }
4744 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4745 {
4746   \pgfpointanchor
4747   {
4748     \@@_env:
4749     - \int_use:N \l_@@_final_i_int
4750     - \int_use:N \l_@@_final_j_int
4751   }
4752   { #1 }
4753   \@@_set_final_coords:
4754 }
4755 \cs_new_protected:Npn \@@_open_x_initial_dim:
4756 {
4757   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4758   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4759   {
4760     \cs_if_exist:cT

```

```

4761     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4762     {
4763       \pgfpointanchor
4764       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4765       { west }
4766       \dim_set:Nn \l_@@_x_initial_dim
4767       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4768     }
4769   }

```

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

```

4770   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4771   {
4772     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4773     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4774     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4775   }
4776 }

4777 \cs_new_protected:Npn \@@_open_x_final_dim:
4778 {
4779   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4780   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4781   {
4782     \cs_if_exist:cT
4783     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4784     {
4785       \pgfpointanchor
4786       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4787       { east }
4788       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4789       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4790     }
4791   }

```

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

```

4792   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4793   {
4794     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4795     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4796     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4797   }
4798 }

```

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.

```

4799 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4800 {
4801   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4802   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4803   {
4804     \@@_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.

```

4805   \bool_if:NT \g_@@_aux_found_bool
4806   {
4807     {
4808       \@@_open_shorten:
4809       \int_if_zero:nTF { #1 }
4810       { \color { nicematrix-first-row } }
4811     }

```

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.

```

4812         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4813         { \color { nicematrix-last-row } }
4814     }
4815     \keys_set:nn { nicematrix / xdots } { #3 }
4816     \@@_color:o \l_@@_xdots_color_tl
4817     \@@_actually_draw_Ldots:
4818 }
4819 }
4820 }
4821 }

```

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`.

```

4822 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4823 {
4824     \bool_if:NTF \l_@@_initial_open_bool
4825     {
4826         \@@_open_x_initial_dim:
4827         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4828         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4829     }
4830     { \@@_set_initial_coords_from_anchor:n { base~east } }
4831     \bool_if:NTF \l_@@_final_open_bool
4832     {
4833         \@@_open_x_final_dim:
4834         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4835         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4836     }
4837     { \@@_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.

```

4838     \bool_lazy_all:nTF
4839     {
4840         \l_@@_initial_open_bool
4841         \l_@@_final_open_bool
4842         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4843     }
4844     {
4845         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4846         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4847     }

```

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

```

4848     {
4849         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4850         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4851     }
4852     \@@_draw_line:
4853 }

```

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.

```

4854 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4855 {
4856   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4857   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4858   {
4859     \@@_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.

```

4860     \bool_if:NT \g_@@_aux_found_bool
4861     {
4862       {
4863         \@@_open_shorten:
4864         \int_if_zero:nTF { #1 }
4865         { \color { nicematrix-first-row } }
4866         {

```

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.

```

4867         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4868         { \color { nicematrix-last-row } }
4869       }
4870       \keys_set:nn { nicematrix / xdots } { #3 }
4871       \@@_color:o \l_@@_xdots_color_tl
4872       \@@_actually_draw_Cdots:
4873     }
4874   }
4875 }
4876 }

```

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`.

```

4877 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4878 {
4879   \bool_if:NTF \l_@@_initial_open_bool
4880   \@@_open_x_initial_dim:
4881   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4882   \bool_if:NTF \l_@@_final_open_bool
4883   \@@_open_x_final_dim:
4884   { \@@_set_final_coords_from_anchor:n { mid-west } }
4885   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4886   {
4887     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4888     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4889     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4890     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4891     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4892   }
4893   {
4894     \bool_if:NT \l_@@_initial_open_bool
4895     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4896     \bool_if:NT \l_@@_final_open_bool

```

```

4897         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4898     }
4899     \@@_draw_line:
4900 }
4901 \cs_new_protected:Npn \@@_open_y_initial_dim:
4902 {
4903     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4904     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4905     {
4906         \cs_if_exist:cT
4907         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4908         {
4909             \pgfpointanchor
4910             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4911             { north }
4912             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4913             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4914         }
4915     }
4916     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4917     {
4918         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4919         \dim_set:Nn \l_@@_y_initial_dim
4920         {
4921             \fp_to_dim:n
4922             {
4923                 \pgf@y
4924                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4925             }
4926         }
4927     }
4928 }
4929 \cs_new_protected:Npn \@@_open_y_final_dim:
4930 {
4931     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4932     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4933     {
4934         \cs_if_exist:cT
4935         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4936         {
4937             \pgfpointanchor
4938             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4939             { south }
4940             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4941             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4942         }
4943     }
4944     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4945     {
4946         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4947         \dim_set:Nn \l_@@_y_final_dim
4948         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4949     }
4950 }

```

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.

```

4951 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4952 {
4953     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4954     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4955     {
4956         \@@_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.

```

4957     \bool_if:NT \g_@@_aux_found_bool
4958     {
4959         {
4960             \@@_open_shorten:
4961             \int_if_zero:nTF { #2 }
4962                 { \color { nicematrix-first-col } }
4963                 {
4964                     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4965                     { \color { nicematrix-last-col } }
4966                 }
4967             \keys_set:nn { nicematrix / xdots } { #3 }
4968             \@@_color:o \l_@@_xdots_color_tl
4969             \bool_if:NTF \l_@@_Vbrace_bool
4970                 \@@_actually_draw_Vbrace:
4971                 \@@_actually_draw_Vdots:
4972         }
4973     }
4974 }
4975 }

```

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`.

```

4976 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4977 {
4978     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4979     \@@_actually_draw_Vdots_i:
4980     \@@_actually_draw_Vdots_ii:
4981     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4982     \@@_draw_line:
4983 }

```

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

```

4984 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4985 {
4986     \@@_open_y_initial_dim:
4987     \@@_open_y_final_dim:
4988     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4989     {
4990         \@@_qpoint:n { col - 1 }
4991         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4992         \dim_sub:Nn \l_@@_x_initial_dim
4993         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4994     }
4995     {
4996         \bool_lazy_and:nnTF
4997         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4998         { \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”.

```

4999      {
5000          \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5001          \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5002          \dim_add:Nn \l_@@_x_initial_dim
5003              { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
5004      }

```

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

```

5005      {
5006          \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5007          \dim_set_eq:NN \l_tmpa_dim \pgf@x
5008          \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
5009          \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
5010      }
5011  }
5012 }

```

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.

```

5013 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
5014 {
5015     \bool_set_false:N \l_tmpa_bool
5016     \bool_if:NF \l_@@_initial_open_bool
5017     {
5018         \bool_if:NF \l_@@_final_open_bool
5019         {
5020             \@@_set_initial_coords_from_anchor:n { south-west }
5021             \@@_set_final_coords_from_anchor:n { north-west }
5022             \bool_set:Nn \l_tmpa_bool
5023                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
5024         }
5025     }

```

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

```

5026     \bool_if:NTF \l_@@_initial_open_bool
5027     {
5028         \@@_open_y_initial_dim:
5029         \@@_set_final_coords_from_anchor:n { north }
5030         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
5031     }
5032     {
5033         \@@_set_initial_coords_from_anchor:n { south }
5034         \bool_if:NTF \l_@@_final_open_bool
5035             \@@_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.

```

5036     {
5037         \@@_set_final_coords_from_anchor:n { north }
5038         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
5039         {
5040             \dim_set:Nn \l_@@_x_initial_dim
5041             {
5042                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
5043                     \l_@@_x_initial_dim \l_@@_x_final_dim
5044             }
5045         }
5046     }
5047 }
5048 }

```

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`.

```

5049 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
5050 {
5051   \bool_if:NTF \l_@@_initial_open_bool
5052     \@@_open_y_initial_dim:
5053     { \@@_set_initial_coords_from_anchor:n { south } }
5054   \bool_if:NTF \l_@@_final_open_bool
5055     \@@_open_y_final_dim:
5056     { \@@_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.

```

5057   \int_if_zero:nTF \l_@@_initial_j_int
5058   {
5059     \@@_qpoint:n { col - 1 }
5060     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5061     \dim_sub:Nn \l_@@_x_initial_dim
5062     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
5063   }

```

Elsewhere, the brace must be drawn left flush.

```

5064   {
5065     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5066     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5067     \dim_add:Nn \l_@@_x_initial_dim
5068     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
5069   }

```

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

```

5070   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5071   \@@_draw_line:
5072 }

```

```

5073 \cs_new:Npn \@@_colsep:
5074 { \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.

```

5075 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
5076 {
5077   \@@_adjust_to_submatrix:nn { #1 } { #2 }
5078   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5079   {
5080     \@@_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.

```

5081     \bool_if:NT \g_@@_aux_found_bool
5082     {
5083     {
5084         \@@_open_shorten:
5085         \keys_set:nn { nicematrix / xdots } { #3 }
5086         \@@_color:o \l_@@_xdots_color_tl
5087         \@@_actually_draw_Ddots:
5088     }
5089     }
5090 }
5091 }
```

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`.

```

5092 \cs_new_protected:Npn \@@_actually_draw_Ddots:
5093 {
5094     \bool_if:NTF \l_@@_initial_open_bool
5095     {
5096         \@@_open_y_initial_dim:
5097         \@@_open_x_initial_dim:
5098     }
5099     { \@@_set_initial_coords_from_anchor:n { south~east } }
5100     \bool_if:NTF \l_@@_final_open_bool
5101     {
5102         \@@_open_x_final_dim:
5103         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5104     }
5105     { \@@_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.

```

5106     \bool_if:NT \l_@@_parallelize_diags_bool
5107     {
5108         \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).

```

5109         \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.

```

5110     {
5111         \dim_gset:Nn \g_@@_delta_x_one_dim
5112         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5113         \dim_gset:Nn \g_@@_delta_y_one_dim
5114         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5115     }
```

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`.

```

5116     {
5117         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
5118         {
5119             \dim_set:Nn \l_@@_y_final_dim
5120             {
5121                 \l_@@_y_initial_dim +
5122                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5123                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
5124             }
5125         }
5126     }
5127 }
5128 \@@_draw_line:
5129 }

```

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.

```

5130 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
5131 {
5132     \@@_adjust_to_submatrix:nn { #1 } { #2 }
5133     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5134     {
5135         \@@_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.

```

5136         \bool_if:NT \g_@@_aux_found_bool
5137         {
5138             {
5139                 \@@_open_shorten:
5140                 \keys_set:nn { nicematrix / xdots } { #3 }
5141                 \@@_color:o \l_@@_xdots_color_tl
5142                 \@@_actually_draw_Iddots:
5143             }
5144         }
5145     }
5146 }

```

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`.

```

5147 \cs_new_protected:Npn \@@_actually_draw_Iddots:
5148 {
5149     \bool_if:NTF \l_@@_initial_open_bool
5150     {
5151         \@@_open_y_initial_dim:
5152         \@@_open_x_initial_dim:
5153     }
5154     { \@@_set_initial_coords_from_anchor:n { south-west } }
5155     \bool_if:NTF \l_@@_final_open_bool

```

```

5156 {
5157   \l_@@_open_y_final_dim:
5158   \l_@@_open_x_final_dim:
5159 }
5160 { \l_@@_set_final_coords_from_anchor:n { north-east } }
5161 \bool_if:NT \l_@@_parallelize_diags_bool
5162 {
5163   \int_gincr:N \g_@@_iddots_int
5164   \int_compare:nNnTF \g_@@_iddots_int = 1
5165   {
5166     \dim_gset:Nn \g_@@_delta_x_two_dim
5167       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5168     \dim_gset:Nn \g_@@_delta_y_two_dim
5169       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5170   }
5171   {
5172     \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
5173     {
5174       \dim_set:Nn \l_@@_y_final_dim
5175       {
5176         \l_@@_y_initial_dim +
5177         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5178         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
5179       }
5180     }
5181   }
5182 }
5183 \l_@@_draw_line:
5184 }

```

17 The actual instructions for drawing the dotted lines with TikZ

The command `\l_@@_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`

```

5185 \cs_new_protected:Npn \l_@@_draw_line:
5186 {
5187   \pgfrememberpicturepositiononpagetrue
5188   \pgf@relevantforpicturesizefalse
5189   \bool_lazy_or:nnTF
5190     \l_@@_dotted_bool
5191     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
5192     \l_@@_draw_standard_dotted_line:
5193     \l_@@_draw_unstandard_dotted_line:
5194 }

```

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.

```

5195 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
5196 {
5197   \begin { scope }
5198   \@@_draw_unstandard_dotted_line:o
5199   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
5200 }

```

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.

```

5201 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
5202 {
5203   \@@_draw_unstandard_dotted_line:nnnn #1
5204   { #1 }
5205   \l_@@_xdots_up_tl
5206   \l_@@_xdots_down_tl
5207   \l_@@_xdots_middle_tl
5208 }
5209 \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.

```

5210 \AtBeginDocument
5211 {
5212   \IfPackageLoadedT { tikz }
5213   {
5214     \tikzset
5215     {
5216       @@_node_above / .style = { sloped , above } ,
5217       @@_node_below / .style = { sloped , below } ,
5218       @@_node_middle / .style =
5219       {
5220         sloped ,
5221         inner~sep = \c_@@_innersep_middle_dim
5222       }
5223     }
5224   }
5225 }

5226 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
5227 {

```

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.

```

5228   \dim_zero_new:N \l_@@_l_dim
5229   \dim_set:Nn \l_@@_l_dim
5230   {
5231     \fp_to_dim:n
5232     {
5233       sqrt
5234       (
5235         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5236         +
5237         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5238       )
5239     }
5240   }

```

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.

```

5241 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5242 {
5243   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5244   \@@_draw_unstandard_dotted_line_i:
5245 }

```

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

```

5246 \bool_if:NT \l_@@_xdots_h_labels_bool
5247 {
5248   \tikzset
5249   {
5250     @@_node_above / .style = { auto = left } ,
5251     @@_node_below / .style = { auto = right } ,
5252     @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
5253   }
5254 }
5255 \tl_if_empty:nF { #4 }
5256 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
5257 \dim_zero:N \l_tmpa_dim
5258 \dim_zero:N \l_tmpb_dim
5259 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
5260 {

```

We test whether the brace is vertical or horizontal.

```

5261 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5262 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
5263 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5264 }
5265 {
5266   \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5267   {
5268     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5269     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5270     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5271   }
5272 }
5273 \use:e
5274 {
5275   \exp_not:N \begin { scope }
5276   [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5277 }
5278 \draw
5279 [ #1 ]
5280 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5281 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
5282 node [ @@_node_below ] { $ \scriptstyle #3 $ }
5283 node [ @@_node_above ] { $ \scriptstyle #2 $ }
5284 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5285 \end { scope }
5286 \end { scope }
5287 }
5288 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5289 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5290 {
5291   \dim_set:Nn \l_tmpa_dim
5292   {
5293     \l_@@_x_initial_dim
5294     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5295     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

5296     }
5297     \dim_set:Nn \l_tmpb_dim
5298     {
5299         \l_@@_y_initial_dim
5300         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5301         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5302     }
5303     \dim_set:Nn \l_@@_tmpc_dim
5304     {
5305         \l_@@_x_final_dim
5306         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5307         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5308     }
5309     \dim_set:Nn \l_@@_tmpd_dim
5310     {
5311         \l_@@_y_final_dim
5312         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5313         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5314     }
5315     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5316     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5317     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5318     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5319 }

```

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

```

5320 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5321 {
5322 {

```

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.

```

5323     \dim_zero_new:N \l_@@_l_dim
5324     \dim_set:Nn \l_@@_l_dim
5325     {
5326         \fp_to_dim:n
5327         {
5328             sqrt
5329             (
5330                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5331                 +
5332                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5333             )
5334         }
5335     }

```

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.

```

5336     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5337     {
5338         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5339         \@@_draw_standard_dotted_line_i:
5340     }
5341 }
5342 \bool_lazy_all:nF
5343 {
5344     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5345     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5346     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5347 }

```

```

5348 \@@_labels_standard_dotted_line:
5349 }
5350 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5351 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5352 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

5353 \int_set:Nn \l_1tmpa_int
5354 {
5355 \dim_ratio:nn
5356 {
5357 \l_@@_l_dim
5358 - \l_@@_xdots_shorten_start_dim
5359 - \l_@@_xdots_shorten_end_dim
5360 }
5361 \l_@@_xdots_inter_dim
5362 }

```

The dimensions $\l_1\text{tmpa_dim}$ and $\l_1\text{tmpb_dim}$ are the coordinates of the vector between two dots in the dotted line.

```

5363 \dim_set:Nn \l_1tmpa_dim
5364 {
5365 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5366 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5367 }
5368 \dim_set:Nn \l_1tmpb_dim
5369 {
5370 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5371 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5372 }

```

In the loop over the dots, the dimensions $\l_1\text{@@_x_initial_dim}$ and $\l_1\text{@@_y_initial_dim}$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5373 \dim_gadd:Nn \l_@@_x_initial_dim
5374 {
5375 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5376 \dim_ratio:nn
5377 {
5378 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5379 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5380 }
5381 { 2 \l_@@_l_dim }
5382 }
5383 \dim_gadd:Nn \l_@@_y_initial_dim
5384 {
5385 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5386 \dim_ratio:nn
5387 {
5388 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5389 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5390 }
5391 { 2 \l_@@_l_dim }
5392 }
5393 \pgf@relevantforpicturesizefalse
5394 \int_step_inline:nnn \c_zero_int \l_1tmpa_int
5395 {
5396 \pgfpathcircle
5397 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5398 \l_@@_xdots_radius_dim
5399 \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
5400 \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
5401 }
5402 \pgfusepathqfill
5403 }

```

```

5404 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5405 {
5406   \pgfscope
5407   \pgftransformshift
5408   {
5409     \pgfpointlineatime { 0.5 }
5410     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5411     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5412   }
5413   \fp_set:Nn \l_tmpa_fp
5414   {
5415     atand
5416     (
5417       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5418       \l_@@_x_final_dim - \l_@@_x_initial_dim
5419     )
5420   }
5421   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5422   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5423   \tl_if_empty:NF \l_@@_xdots_middle_tl
5424   {
5425     \begin { pgfscope }
5426     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5427     \pgfnode
5428     { rectangle }
5429     { center }
5430     {
5431       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5432       {
5433         $ % $
5434         \scriptstyle \l_@@_xdots_middle_tl
5435         $ % $
5436       }
5437     }
5438     { }
5439     {
5440       \pgfsetfillcolor { white }
5441       \pgfusepath { fill }
5442     }
5443     \end { pgfscope }
5444   }
5445   \tl_if_empty:NF \l_@@_xdots_up_tl
5446   {
5447     \pgfnode
5448     { rectangle }
5449     { south }
5450     {
5451       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5452       {
5453         $ % $
5454         \scriptstyle \l_@@_xdots_up_tl
5455         $ % $
5456       }
5457     }
5458     { }
5459     { \pgfusepath { } }
5460   }
5461   \tl_if_empty:NF \l_@@_xdots_down_tl
5462   {
5463     \pgfnode
5464     { rectangle }
5465     { north }
5466     {

```

```

5467         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5468         {
5469             $ % $
5470             \scriptstyle \l_@@_xdots_down_tl
5471             $ % $
5472         }
5473     }
5474     { }
5475     { \pgfusepath { } }
5476 }
5477 \endpgfscope
5478 }

```

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.

```

5479 \AtBeginDocument
5480 {

```

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

```

5481     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } { } } }
5482     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5483     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5484     {
5485         \int_if_zero:nTF \c@jCol
5486         { \@@_error:nn { in~first~col } { \Ldots } }
5487         {
5488             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5489             { \@@_error:nn { in~last~col } { \Ldots } }
5490             {
5491                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5492                 { #1 , down = #2 , up = #3 , middle = #4 }
5493             }
5494         }
5495         \bool_if:NF \l_@@_nullify_dots_bool
5496         { \phantom { \ensuremath { \@@_old_ldots: } } }
5497         \bool_gset_true:N \g_@@_empty_cell_bool
5498     }

5499     \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5500     \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5501     {
5502         \int_if_zero:nTF \c@jCol
5503         { \@@_error:nn { in~first~col } { \Cdots } }
5504         {
5505             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5506             { \@@_error:nn { in~last~col } { \Cdots } }
5507             {
5508                 \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }

```

```

5509         { #1 , down = #2 , up = #3 , middle = #4 }
5510     }
5511 }
5512 \bool_if:NF \l_@@_nullify_dots_bool
5513 { \phantom { \ensuremath { \@@_old_cdots: } } }
5514 \bool_gset_true:N \g_@@_empty_cell_bool
5515 }

5516 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5517 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5518 {
5519     \int_if_zero:nTF \c@iRow
5520     { \@@_error:nn { in~first~row } { \Vdots } }
5521     {
5522         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5523         { \@@_error:nn { in~last~row } { \Vdots } }
5524         {
5525             \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5526             { #1 , down = #2 , up = #3 , middle = #4 }
5527         }
5528     }
5529     \bool_if:NF \l_@@_nullify_dots_bool
5530     { \phantom { \ensuremath { \@@_old_vdots: } } }
5531     \bool_gset_true:N \g_@@_empty_cell_bool
5532 }

5533 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5534 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5535 {
5536     \int_case:nnF \c@iRow
5537     {
5538         0 { \@@_error:nn { in~first~row } { \Ddots } }
5539         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5540     }
5541     {
5542         \int_case:nnF \c@jCol
5543         {
5544             0 { \@@_error:nn { in~first~col } { \Ddots } }
5545             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5546         }
5547         {
5548             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5549             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5550             { #1 , down = #2 , up = #3 , middle = #4 }
5551         }
5552     }
5553 }
5554 \bool_if:NF \l_@@_nullify_dots_bool
5555 { \phantom { \ensuremath { \@@_old_ddots: } } }
5556 \bool_gset_true:N \g_@@_empty_cell_bool
5557 }

5558 \cs_new_protected:Npn \@@_Iddots:
5559 { \@@_collect_options:n { \@@_Iddots_i } }
5560 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5561 {
5562     \int_case:nnF \c@iRow
5563     {
5564         0 { \@@_error:nn { in~first~row } { \Iddots } }
5565         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5566     }

```

```

5567     {
5568         \int_case:nnF \c@jCol
5569         {
5570             0 { \@@_error:nn { in~first~col } { \Iddots } }
5571             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5572         }
5573         {
5574             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5575             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5576             { #1 , down = #2 , up = #3 , middle = #4 }
5577         }
5578     }
5579     \bool_if:NF \l_@@_nullify_dots_bool
5580     { \phantom { \ensuremath { \@@_old_iddots: } } }
5581     \bool_gset_true:N \g_@@_empty_cell_bool
5582 }
5583 }

```

End of the \AddToHook.

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

```

5584 \keys_define:nn { nicematrix / Ddots }
5585 {
5586     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5587     draw-first .value_forbidden:n = true ,
5588 }

```

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

```

5589 \cs_new_protected:Npn \@@_Hspace:
5590 {
5591     \bool_gset_true:N \g_@@_empty_cell_bool
5592     \hspace
5593 }

```

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.

```

5594 \cs_new:Npn \@@_Hdotsfor:
5595 {
5596     \bool_lazy_and:nnTF
5597     { \int_if_zero_p:n \c@jCol }
5598     { \int_if_zero_p:n \l_@@_first_col_int }
5599     {
5600         \bool_if:NTF \g_@@_after_col_zero_bool
5601         {
5602             \multicolumn { 1 } { c } { }
5603             \@@_Hdotsfor_i:
5604         }
5605         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5606     }
5607     {
5608         \multicolumn { 1 } { c } { }
5609         \@@_Hdotsfor_i:
5610     }
5611 }

```

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:).

```

5612 \AtBeginDocument
5613 {

```

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

```
5614 \cs_new_protected:Npn \@@_Hdotsfor_i:
5615 { \@@_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).

```
5616 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5617 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5618 {
5619   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5620   {
5621     \@@_Hdotsfor:nnnn
5622     { \int_use:N \c@iRow }
5623     { \int_use:N \c@jCol }
5624     { #2 }
5625     {
5626       #1 , #3 ,
5627       down = \exp_not:n { #4 } ,
5628       up = \exp_not:n { #5 } ,
5629       middle = \exp_not:n { #6 }
5630     }
5631   }
5632   \prg_replicate:nn { #2 - 1 }
5633   {
5634     &
5635     \multicolumn { 1 } { c } { }
5636     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5637   }
5638 }
5639 }
```

```
5640 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5641 {
5642   \bool_set_false:N \l_@@_initial_open_bool
5643   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5644 \int_set:Nn \l_@@_initial_i_int { #1 }
5645 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5646 \int_compare:nNnTF { #2 } = 1
5647 {
5648   \int_set:Nn \l_@@_initial_j_int 1
5649   \bool_set_true:N \l_@@_initial_open_bool
5650 }
5651 {
5652   \cs_if_exist:cTF
5653   {
5654     pgf @ sh @ ns @ \@@_env:
5655     - \int_use:N \l_@@_initial_i_int
5656     - \int_eval:n { #2 - 1 }
5657   }
5658   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5659   {
5660     \int_set:Nn \l_@@_initial_j_int { #2 }
5661     \bool_set_true:N \l_@@_initial_open_bool
5662   }
5663 }
5664 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5665 {
5666   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5667   \bool_set_true:N \l_@@_final_open_bool
```

```

5668 }
5669 {
5670   \cs_if_exist:cTF
5671   {
5672     pgf @ sh @ ns @ \@@_env:
5673     - \int_use:N \l_@@_final_i_int
5674     - \int_eval:n { #2 + #3 }
5675   }
5676   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5677   {
5678     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5679     \bool_set_true:N \l_@@_final_open_bool
5680   }
5681 }
5682 \bool_if:NT \g_@@_aux_found_bool
5683 {
5684   {
5685     \@@_open_shorten:
5686     \int_if_zero:nTF { #1 }
5687     { \color { nicematrix-first-row } }
5688     {
5689       \int_compare:nNt { #1 } = \g_@@_row_total_int
5690       { \color { nicematrix-last-row } }
5691     }
5692     \keys_set:nn { nicematrix / xdots } { #4 }
5693     \@@_color:o \l_@@_xdots_color_tl
5694     \@@_actually_draw_Ldots:
5695   }
5696 }

```

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).

```

5697   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5698   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5699 }

```

```

5700 \AtBeginDocument
5701 {
5702   \cs_new_protected:Npn \@@_Vdotsfor:
5703   { \@@_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`).

```

5704   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5705   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5706   {
5707     \bool_gset_true:N \g_@@_empty_cell_bool
5708     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5709     {
5710       \@@_Vdotsfor:nnnn
5711       { \int_use:N \c@iRow }
5712       { \int_use:N \c@jCol }
5713       { #2 }
5714       {
5715         #1 , #3 ,
5716         down = \exp_not:n { #4 } ,
5717         up = \exp_not:n { #5 } ,
5718         middle = \exp_not:n { #6 }
5719       }
5720     }
5721   }
5722 }

```

#1 is the number of row;
#2 is the number of column;
#3 is the numbers of rows which are involved;

```

5723 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5724 {
5725   \bool_set_false:N \l_@@_initial_open_bool
5726   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5727   \int_set:Nn \l_@@_initial_j_int { #2 }
5728   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5729   \int_compare:nNnTF { #1 } = 1
5730   {
5731     \int_set:Nn \l_@@_initial_i_int 1
5732     \bool_set_true:N \l_@@_initial_open_bool
5733   }
5734   {
5735     \cs_if_exist:cTF
5736     {
5737       pgf @ sh @ ns @ \@@_env:
5738       - \int_eval:n { #1 - 1 }
5739       - \int_use:N \l_@@_initial_j_int
5740     }
5741     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5742     {
5743       \int_set:Nn \l_@@_initial_i_int { #1 }
5744       \bool_set_true:N \l_@@_initial_open_bool
5745     }
5746   }
5747   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5748   {
5749     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5750     \bool_set_true:N \l_@@_final_open_bool
5751   }
5752   {
5753     \cs_if_exist:cTF
5754     {
5755       pgf @ sh @ ns @ \@@_env:
5756       - \int_eval:n { #1 + #3 }
5757       - \int_use:N \l_@@_final_j_int
5758     }
5759     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5760     {
5761       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5762       \bool_set_true:N \l_@@_final_open_bool
5763     }
5764   }
5765   \bool_if:NT \g_@@_aux_found_bool
5766   {
5767     {
5768       \@@_open_shorten:
5769       \int_if_zero:nTF { #2 }
5770       { \color { nicematrix-first-col } }
5771       {
5772         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5773         { \color { nicematrix-last-col } }
5774       }
5775       \keys_set:nn { nicematrix / xdots } { #4 }
5776       \@@_color:o \l_@@_xdots_color_tl
5777       \bool_if:NTF \l_@@_Vbrace_bool
5778       \@@_actually_draw_Vbrace:

```

```

5779         \@@_actually_draw_Vdots:
5780     }
5781 }

```

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).

```

5782     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5783     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5784 }

```

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

```

5785 \NewDocumentCommand \@@_rotate: { 0 { } }
5786 {
5787     \bool_gset_true:N \g_@@_rotate_bool
5788     \keys_set:nn { nicematrix / rotate } { #1 }
5789     \ignorespaces
5790 }

```

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

```

5791 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate-in-p-col } }

5792 \keys_define:nn { nicematrix / rotate }
5793 {
5794     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5795     c .value_forbidden:n = true ,
5796     -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5797     -90 .value_forbidden:n = true ,
5798     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5799 }

```

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).¹⁵

```

5800 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5801 {
5802     \tl_if_empty:nTF { #2 }
5803     { #1 }
5804     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5805 }
5806 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5807 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

¹⁵Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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

```
5808 \AtBeginDocument
5809 {
```

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

```
5810 \tl_set_rescan:Nnn \l_tmpa_tl { }
5811 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5812 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5813 {
5814 {
5815 \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5816 \@@_color:o \l_@@_xdots_color_tl
5817 \use:e
5818 {
5819 \@@_line_i:nn
5820 { \@@_double_int_eval:n #2 - \q_stop }
5821 { \@@_double_int_eval:n #3 - \q_stop }
5822 }
5823 }
5824 }
5825 }

5826 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5827 {
5828 \bool_set_false:N \l_@@_initial_open_bool
5829 \bool_set_false:N \l_@@_final_open_bool
5830 \bool_lazy_or:nnTF
5831 { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5832 { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5833 { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```
5834 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5835 }

5836 \AtBeginDocument
5837 {
5838 \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5839 {
```

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:`.

```
5840 \c_@@_pgfortikzpicture_tl
5841 \@@_draw_line_iii:nn { #1 } { #2 }
5842 \c_@@_endpgfortikzpicture_tl
5843 }
5844 }
```

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

```
5845 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5846 {
5847 \pgfrememberpicturepositiononpagetrue
5848 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5849 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5850 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5851 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5852 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5853 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5854 \@@_draw_line:
5855 }
```

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

20 The command `\RowStyle`

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

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

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

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

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

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

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

```
5856 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNt \c@iRow < }
5857 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNf \c@jCol < }
```

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

```
5858 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5859 {
5860   \tl_gput_right:Ne \g_@@_row_style_tl
5861   {
```

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

```
5862   \exp_not:N
5863   \@@_if_row_less_than:nn
5864   { \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`).

```
5865   {
5866     \exp_not:N
5867     \@@_if_col_greater_than:nn
5868     { \int_use:N \c@jCol }
5869     { \exp_not:n { #1 } \scan_stop: }
5870   }
5871 }
5872 }
5873 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5874 \keys_define:nn { nicematrix / RowStyle }
5875 {
5876   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5877   cell-space-top-limit+ .code:n =
5878     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5879   cell-space-top-limit+ .value_required:n = true ,
5880   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5881   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5882   cell-space-bottom-limit+ .code:n =
5883     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5884   cell-space-bottom-limit+ .value_required:n = true ,
5885   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5886   cell-space-limits .meta:n =
5887     {
5888       cell-space-top-limit = #1 ,
```

```

5889     cell-space-bottom-limit = #1 ,
5890   } ,
5891   cell-space-limits+ .meta:n =
5892   {
5893     cell-space-top-limit += #1 ,
5894     cell-space-bottom-limit += #1 ,
5895   } ,
5896   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5897   color .tl_set:N = \l_@@_color_tl ,
5898   color .value_required:n = true ,
5899   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5900   nb-rows .code:n =
5901     \str_if_eq:eeTF { #1 } { * }
5902     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5903     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5904   nb-rows .value_required:n = true ,
5905   fill .tl_set:N = \l_@@_fill_tl ,
5906   fill .value_required:n = true ,

```

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

```

5907   opacity .tl_set:N = \l_@@_opacity_tl ,
5908   opacity .value_required:n = true ,
5909   rowcolor .tl_set:N = \l_@@_fill_tl ,
5910   rowcolor .value_required:n = true ,
5911   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5912   rounded-corners .default:n = 4 pt ,
5913   unknown .code:n =
5914     \@@_unknown_key:nn
5915     { nicematrix / RowStyle }
5916     { Unknown-key-for-RowStyle }
5917 }

```

```

5918 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5919 {
5920   \group_begin:
5921   \tl_clear:N \l_@@_fill_tl
5922   \tl_clear:N \l_@@_opacity_tl
5923   \tl_clear:N \l_@@_color_tl
5924   \int_set:Nn \l_@@_key_nb_rows_int 1
5925   \dim_zero:N \l_@@_rounded_corners_dim
5926   \dim_zero:N \l_tmpa_dim
5927   \dim_zero:N \l_tmpb_dim
5928   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5929   \tl_if_empty:NF \l_@@_fill_tl
5930   {
5931     \@@_add_opacity_to_fill:
5932     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5933     {

```

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

```

5934     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5935     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5936     {
5937       \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5938       - *
5939     }
5940     { \dim_use:N \l_@@_rounded_corners_dim }
5941   }
5942 }
5943 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5944 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5945 {
5946   \@@_put_in_row_style:e
5947   {
5948     \@@_put_in_cell_after_hook:n
5949     {

```

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

```

5950       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5951       { \dim_use:N \l_tmpa_dim }
5952     }
5953   }
5954 }

```

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

```

5955 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5956 {
5957   \@@_put_in_row_style:e
5958   {
5959     \@@_put_in_cell_after_hook:n
5960     {
5961       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5962       { \dim_use:N \l_tmpb_dim }
5963     }
5964   }
5965 }

```

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

```

5966 \tl_if_empty:NF \l_@@_color_tl
5967 {
5968   \@@_put_in_row_style:e
5969   {
5970     \mode_leave_vertical:
5971     \@@_color:n { \l_@@_color_tl }
5972   }
5973 }

```

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

```

5974 \bool_if:NT \l_@@_bold_row_style_bool
5975 {
5976   \@@_put_in_row_style:n
5977   {
5978     \exp_not:n
5979     {
5980       \if_mode_math:
5981         $ % $
5982         \bfseries \boldmath
5983         $ % $
5984       \else:
5985         \bfseries \boldmath
5986       \fi:
5987     }
5988   }
5989 }
5990 \group_end:
5991 \g_@@_row_style_tl
5992 \ignorespaces
5993 }

```

The following commande must *not* be protected.

```

5994 \cs_new:Npn \@@_rounded_from_row:n #1
5995 {
5996   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5997     { \int_eval:n { #1 } - 1 }
5998     {
5999         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
6000         - \exp_not:n { \int_use:N \c@jCol }
6001     }
6002     { \dim_use:N \l_@@_rounded_corners_dim }
6003 }

```

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).

```

6004 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
6005 {

```

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.

```

6006     \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`.

```

6007     \str_if_in:nnF { #1 } { !! }
6008     {
6009         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

6010         { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
6011     }
6012     \int_if_zero:nTF \l_tmpa_int

```

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

```

6013     {
6014         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
6015         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
6016     }

```

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

```

6017     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
6018 }
6019 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }

```

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

```
6020 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
6021 {
6022   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
6023   {
```

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

```
6024   \group_begin:
6025   \pgfsetcornersarced
6026   { \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.

```
6027   \bool_if:NTF \l_@@_hvlines_bool
6028   {
6029     \pgfpathrectanglecorners
6030     {
6031       \pgfpointadd
6032       { \@@_qpoint:n { row-1 } }
6033       { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
6034     }
6035     {
6036       \pgfpointadd
6037       {
6038         \@@_qpoint:n
6039         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6040       }
6041       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
6042     }
6043   }
6044   {
6045     \pgfpathrectanglecorners
6046     { \@@_qpoint:n { row-1 } }
6047     {
6048       \pgfpointadd
6049       {
6050         \@@_qpoint:n
6051         { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6052       }
6053       { \pgfpoint \c_zero_dim \arrayrulewidth }
6054     }
6055   }
6056   \pgfusepath { clip }
6057   \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```
6058   }
6059 }
```

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

```
6060 \cs_new_protected:Npn \@@_actually_color:
6061 {
6062   \pgfpicture
6063   \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.

```
6064   \@@_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`).

```

6065 \seq_map_indexed_inline:Nn \g_@@_colors_seq
6066 {
6067   \int_compare:nNnTF { ##1 } = 1
6068   {
6069     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
6070     \use:c { g_@@_color _ 1 _tl }
6071     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
6072   }
6073   {
6074     \pgfscope
6075       \@@_color_opacity: ##2
6076       \use:c { g_@@_color _ ##1 _tl }
6077       \tl_gclear:c { g_@@_color _ ##1 _tl }
6078       \pgfusepath { fill }
6079     \endpgfscope
6080   }
6081 }
6082 \endpgfpicture
6083 }

```

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

```

6084 \cs_new_protected:Npn \@@_color_opacity:
6085 {
6086   \peek_meaning:NTF [
6087     \@@_color_opacity:w
6088     { \@@_color_opacity:w [ ] }
6089   }

```

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.

```

6090 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
6091 {
6092   \tl_clear:N \l_tmpa_tl
6093   \keys_set_known:nn { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

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

```

6094   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
6095   \tl_if_empty:NTF \l_tmpb_tl
6096     \@declaredcolor
6097     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
6098 }

```

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

```

6099 \keys_define:nn { nicematrix / color-opacity }
6100 {
6101   opacity .tl_set:N          = \l_tmpa_tl ,
6102   opacity .value_required:n = true
6103 }

```

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

```

6104 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
6105 {
6106   \def \l_@@_rows_tl { #1 }
6107   \def \l_@@_cols_tl { #2 }
6108   \@@_cartesian_path:
6109 }

```

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

```
6110 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
6111 {
6112   \tl_if_blank:nF { #2 }
6113   {
6114     \@@_add_to_colors_seq:en
6115     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6116     { \@@_cartesian_color:nn { #3 } { - } }
6117   }
6118 }
```

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

```
6119 \NewDocumentCommand \@@_columncolor { 0 { } m m }
6120 {
6121   \tl_if_blank:nF { #2 }
6122   {
6123     \@@_add_to_colors_seq:en
6124     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6125     { \@@_cartesian_color:nn { - } { #3 } }
6126   }
6127 }
```

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

```
6128 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
6129 {
6130   \tl_if_blank:nF { #2 }
6131   {
6132     \@@_add_to_colors_seq:en
6133     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6134     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
6135   }
6136 }
```

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

```
6137 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
6138 {
6139   \tl_if_blank:nF { #2 }
6140   {
6141     \@@_add_to_colors_seq:en
6142     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6143     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
6144   }
6145 }
```

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

```
6146 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
6147 {
6148   \@@_cut_on_hyphen:w #1 \q_stop
6149   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6150   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6151   \@@_cut_on_hyphen:w #2 \q_stop
6152   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
6153   \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`.

```
6154 \@@_cartesian_path:n { #3 }
6155 }
```

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

```

6156 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
6157 {
6158   \clist_map_inline:nn { #3 }
6159   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
6160 }

6161 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
6162 {
6163   \int_step_inline:nn \c@iRow
6164   {
6165     \int_step_inline:nn \c@jCol
6166     {
6167       \int_if_even:nTF { ####1 + ##1 }
6168       { \@@_cellcolor [ #1 ] { #2 } }
6169       { \@@_cellcolor [ #1 ] { #3 } }
6170       { ##1 - ####1 }
6171     }
6172   }
6173 }

```

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”.

```

6174 \NewDocumentCommand \@@_arraycolor { 0 { } m }
6175 {
6176   \@@_rectanglecolor [ #1 ] { #2 }
6177   { 1 - 1 }
6178   { \int_use:N \c@iRow - \int_use:N \c@jCol }
6179 }

6180 \keys_define:nn { nicematrix / rowcolors }
6181 {
6182   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
6183   cols .tl_set:N = \l_@@_cols_tl ,
6184   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
6185   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
6186 }

```

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*.

```

6187 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
6188 {

```

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

```

6189   \seq_clear_new:N \l_@@_colors_seq
6190   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
6191   \tl_clear_new:N \l_@@_cols_tl
6192   \tl_set:Nn \l_@@_cols_tl { - }
6193   \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).

```

6194   \int_zero_new:N \l_@@_color_int
6195   \int_set:Nn \l_@@_color_int 1
6196   \bool_if:NT \l_@@_respect_blocks_bool
6197   {

```

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`).

```

6198     \seq_set_filter:NNn \l_tmpa_seq \g_@@_pos_of_blocks_seq
6199     { \@@_not_in_exterior_p:nnnnn ##1 }
6200 }

```

#2 is the list of intervals of rows.

```

6201     \clist_map_inline:nn { #2 }
6202     {
6203         \tl_set:Nn \l_tmpa_tl { ##1 }
6204         \tl_if_in:NnTF \l_tmpa_tl { - }
6205         { \@@_cut_on_hyphen:w ##1 \q_stop }
6206         { \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.

```

6207         \int_set:Nn \l_tmpa_int \l_tmpa_tl
6208         \int_set:Nn \l_@@_color_int
6209         { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
6210         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
6211         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
6212         {

```

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

```

6213             \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).

```

6214             \bool_if:NT \l_@@_respect_blocks_bool
6215             {
6216                 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
6217                 { \@@_intersect_our_row_p:nnnnn #####1 }
6218                 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

6219             }
6220             \tl_set:Nn \l_@@_rows_tl
6221             { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

6222             \tl_set:Nn \l_@@_color_tl
6223             {
6224                 \@@_color_index:n
6225                 {
6226                     \int_mod:nn
6227                     { \l_@@_color_int - 1 }
6228                     { \seq_count:N \l_@@_colors_seq }
6229                     + 1
6230                 }
6231             }
6232             \tl_if_empty:NF \l_@@_color_tl
6233             {
6234                 \use:e
6235                 {
6236                     \@@_add_to_colors_seq:nn
6237                     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
6238                     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
6239                 }
6240             }
6241             \int_incr:N \l_@@_color_int
6242             \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
6243         }
6244     }
6245 }

```

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.

```
6246 \cs_new:Npn \@@_color_index:n #1
6247 {
```

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

```
6248 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
6249 { \@@_color_index:n { #1 - 1 } }
6250 { \seq_item:Nn \l_@@_colors_seq { #1 } }
6251 }
```

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

```
6252 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
6253 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

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

```
6254 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
6255 {
6256 \int_compare:nNt { #3 } > \l_tmpb_int
6257 { \int_set:Nn \l_tmpb_int { #3 } }
6258 }
```

```
6259 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
6260 {
6261 \int_if_zero:nTF { #4 }
6262 \prg_return_false:
6263 {
6264 \int_compare:nNtTF { #2 } > \c@jCol
6265 \prg_return_false:
6266 \prg_return_true:
6267 }
6268 }
```

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

```
6269 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6270 {
6271 \int_compare:nNtTF { #1 } > \l_tmpa_int
6272 \prg_return_false:
6273 {
6274 \int_compare:nNtTF \l_tmpa_int > { #3 }
6275 \prg_return_false:
6276 \prg_return_true:
6277 }
6278 }
```

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`).

```
6279 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6280 {
6281 \dim_compare:nNtTF { #1 } = \c_zero_dim
6282 {
6283 \bool_if:NtF \l_@@_nocolor_used_bool
6284 { \@@_cartesian_path_normal_ii: }
6285 {
```

Even if the user has used the key `corners` the list of cells in the corners may be empty. That's why we test against `\l_@@_corners_cells_clist` and not `\l_@@_corners_clist`.

```

6286         \clist_if_empty:NTF \l_@@_corners_cells_clist
6287         { \@@_cartesian_path_normal_i:n { #1 } }
6288         { \@@_cartesian_path_normal_ii: }
6289     }
6290 }
6291 { \@@_cartesian_path_normal_i:n { #1 } }
6292 }

```

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.

```

6293 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6294 {
6295     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

6296     \clist_map_inline:Nn \l_@@_cols_tl
6297     {

```

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

```

6298         \def \l_tmpa_tl { ##1 }
6299         \tl_if_in:NnTF \l_tmpa_tl { - }
6300         { \@@_cut_on_hyphen:w ##1 \q_stop }
6301         { \def \l_tmpb_tl { ##1 } }
6302         \tl_if_empty:NTF \l_tmpa_tl
6303         { \def \l_tmpa_tl { 1 } }
6304         {
6305             \str_if_eq:eeT \l_tmpa_tl { * }
6306             { \def \l_tmpa_tl { 1 } }
6307         }
6308         \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6309         { \@@_error:n { Invalid~col~number } }
6310         \tl_if_empty:NTF \l_tmpb_tl
6311         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6312         {
6313             \str_if_eq:eeT \l_tmpb_tl { * }
6314             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6315         }
6316         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6317         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6318         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6319         \@@_qpoint:n { col - \l_tmpa_tl }
6320         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6321         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6322         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6323         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6324         \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.

```

6325     \clist_map_inline:Nn \l_@@_rows_tl
6326     {
6327         \def \l_tmpa_tl { #####1 }
6328         \tl_if_in:NnTF \l_tmpa_tl { - }
6329         { \@@_cut_on_hyphen:w #####1 \q_stop }
6330         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6331         \tl_if_empty:NTF \l_tmpa_tl
6332         { \def \l_tmpa_tl { 1 } }
6333         {
6334             \str_if_eq:eeT \l_tmpa_tl { * }
6335             { \def \l_tmpa_tl { 1 } }
6336         }

```

```

6337 \tl_if_empty:NTF \l_tmpb_tl
6338 { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6339 {
6340   \str_if_eq:eeT \l_tmpb_tl { * }
6341   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6342 }
6343 \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6344 { \@@_error:n { Invalid~row~number } }
6345 \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6346 { \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`.

```

6347 \cs_if_exist:cF
6348 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6349 {
6350   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6351   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6352   \@@_qpoint:n { row - \l_tmpa_tl }
6353   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6354   \pgfpathrectanglecorners
6355     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6356     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6357 }
6358 }
6359 }
6360 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6361 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6362 {
6363   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6364   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6365 \clist_map_inline:Nn \l_@@_cols_tl
6366 {
6367   \@@_qpoint:n { col - ##1 }
6368   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6369     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6370     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6371   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6372   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6373 \clist_map_inline:Nn \l_@@_rows_tl
6374 {
6375   \@@_if_in_corner:nF { #####1 - ##1 }
6376   {
6377     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6378     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6379     \@@_qpoint:n { row - #####1 }
6380     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6381     \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6382     {
6383       \pgfpathrectanglecorners
6384         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6385         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6386     }
6387   }
6388 }
6389 }
6390 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor`, etc.

```
6391 \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

```
6392 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6393 {
6394   \bool_set_true:N \l_@@_nocolor_used_bool
6395   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6396   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
6397   \clist_map_inline:Nn \l_@@_rows_tl
6398   {
6399     \clist_map_inline:Nn \l_@@_cols_tl
6400     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6401   }
6402 }
```

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`.

```
6403 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6404 {
6405   \clist_set_eq:NN \l_tmpa_clist #1
6406   \clist_clear:N #1
6407   \clist_map_inline:Nn \l_tmpa_clist
6408   {
```

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

```
6409     \def \l_tmpa_tl { ##1 }
6410     \tl_if_in:NnTF \l_tmpa_tl { - }
6411     { \@@_cut_on_hyphen:w ##1 \q_stop }
6412     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6413     \bool_lazy_or:nnT
6414     { \str_if_eq_p:ee \l_tmpa_tl { * } }
6415     { \tl_if_blank_p:o \l_tmpa_tl }
6416     { \def \l_tmpa_tl { 1 } }
6417     \bool_lazy_or:nnT
6418     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6419     { \tl_if_blank_p:o \l_tmpb_tl }
6420     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6421     \int_compare:nNnT \l_tmpb_tl > { #2 }
6422     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6423     \int_step_tokens:nnn \l_tmpa_tl \l_tmpb_tl { \clist_put_right:Nn #1 }
6424   }
6425 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
6426 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6427 {
6428   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6429   {
```

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`).

```
6430     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6431     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6432   }
6433   \ignorespaces
6434 }
```

```

6435 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6436 { \@@_error:n { cellcolor~in~Block } }
6437 % \end{macrocode}
6438 %
6439 % \begin{macrocode}
6440 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6441 { \@@_error:n { rowcolor~in~Block } }
6442 % \end{macrocode}
6443 %
6444 % \bigskip
6445 % The following command will be linked to |\rowcolor| in the tabular.
6446 % \begin{macrocode}
6447 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6448 {
6449   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6450   {
6451     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6452     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6453     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6454   }
6455   \ignorespaces
6456 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6457 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6458 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6459 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6460 {

```

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`.

```

6461   \seq_gclear:N \g_tmpa_seq
6462   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6463   { \@@_rowlistcolors_tabular:nnnn #1 }
6464   \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`.

```

6465   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6466   {
6467     { \int_use:N \c@iRow }
6468     { \exp_not:n { #1 } }
6469     { \exp_not:n { #2 } }
6470     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6471   }
6472   \ignorespaces
6473 }

```

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`).

```
6474 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6475 {
6476   \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).

```
6477   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6478   {
6479     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6480     {
6481       \@@_rowlistcolors
6482       [ \exp_not:n { #2 } ]
6483       { #1 - \int_eval:n { \c@iRow - 1 } }
6484       { \exp_not:n { #3 } }
6485       [ \exp_not:n { #4 } ]
6486     }
6487   }
6488 }
```

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.

```
6489 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6490 {
6491   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6492   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6493   \seq_gclear:N \g_@@_rowlistcolors_seq
6494 }

6495 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6496 {
6497   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6498   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6499 }
```

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.

```
6500 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6501 {
```

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").

```
6502   \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6503   {
6504     \seq_gput_right:NV \g_@@_columncolor_seq \c@jCol
```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```
6505     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6506     {
6507       \exp_not:N \columncolor [ #1 ]
6508       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6509     }
6510   }
6511 }
```

```

6512 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6513 {
6514   \clist_map_inline:nn { #1 }
6515   {
6516     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6517     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99!
6518     \columncolor { nocolor } { ##1 }
6519   }
6520 }

6521 \cs_new_protected:Npn \@@_EmptyRow:n #1
6522 {
6523   \clist_map_inline:nn { #1 }
6524   {
6525     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6526     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99!
6527     \rowcolor { nocolor } { ##1 }
6528   }
6529 }

```

The following command must *not* be protected.

```

6530 \cs_new:Npn \@@_FalseRow
6531 {
6532   \noalign
6533   {
6534     \skip_vertical:n
6535     { - \box_ht_plus_dp:N \@arstrutbox + \l_@@_width_of_false_dim }

```

We raise `\g_@@_FalseRow_bool` to prevent the application of `cell-space-top-limit` and `cell-space-bottom-limit`. We will revert that global switch right after.

```

6536     \bool_gset_true:N \g_@@_FalseRow_bool
6537   }
6538
6539   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6540   \multicolumn { 1 } { @{} c @{} } { }
6541   \bool_gset_false:N \g_@@_FalseRow_bool

```

We keep in memory a list of the `\FalseRow` because we use it with `create-blocks-in-col`.

```

6542   \clist_gput_right:Ne \g_@@_false_rows_clist { \arabic { iRow } }
6543   \\
6544 }

6545 \cs_new:Npn \@@_FalseRow_in_cell: { \@@_error:n { Misuse-of-FalseRow } }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` 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`.

```

6546 \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`.

```

6547 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6548 {
6549   \int_if_zero:nTF \l_@@_first_col_int
6550     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6551     {
6552       \int_if_zero:nTF \c@jCol
6553         {
6554           \int_compare:nNnF \c@iRow = { -1 }
6555           {
6556             \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6557             { #1 }
6558           }
6559         }
6560       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6561     }
6562 }

```

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.

```

6563 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6564 {
6565   \int_if_zero:nF \c@iRow
6566   {
6567     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6568     {
6569       \int_compare:nNnT \c@jCol > \c_zero_int
6570       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6571     }
6572   }
6573 }

```

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`.

```

6574 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6575 {
6576   \IfPackageLoadedTF { tikz }
6577   {
6578     \IfPackageLoadedTF { booktabs }
6579     { #2 }
6580     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6581   }
6582   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6583 }

6584 \NewExpandableDocumentCommand { \@@_TopRule } { }
6585 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6586 \cs_new:Npn \@@_TopRule_i:
6587 {
6588   \noalign \bgroup
6589   \peek_meaning:NTF [
6590     { \@@_TopRule_ii: }
6591     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6592   }

6593 \NewDocumentCommand \@@_TopRule_ii: { o }
6594 {
6595   \tl_gput_right:Ne \g_@@_rules_tl
6596   {

```

```

6597 \@@_draw_hruler:n
6598 {
6599     position = \int_eval:n { \c@iRow + 1 } ,
6600     tikz =
6601     {
6602         line-width = #1 ,
6603         yshift = 0.25 \arrayrulewidth ,
6604         shorten~< = - 0.5 \arrayrulewidth
6605     } ,
6606     total-width = #1
6607 }
6608 }
6609 \skip_vertical:n { \belowrulesep + #1 }
6610 \egroup
6611 }
6612 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6613 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6614 \cs_new:Npn \@@_BottomRule_i:
6615 {
6616     \noalign \bgroup
6617     \peek_meaning:NTF [
6618         { \@@_BottomRule_ii: }
6619         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6620     ]
6621 }
6622 \NewDocumentCommand \@@_BottomRule_ii: { o }
6623 {
6624     \tl_gput_right:Ne \g_@@_rules_tl
6625     {
6626         \@@_draw_hruler:n
6627         {
6628             position = \int_eval:n { \c@iRow + 1 } ,
6629             tikz =
6630             {
6631                 line-width = #1 ,
6632                 yshift = 0.25 \arrayrulewidth ,
6633                 shorten~< = - 0.5 \arrayrulewidth
6634             } ,
6635             total-width = #1 ,
6636         }
6637         \skip_vertical:N \aboverulesep
6638         \@@_create_row_node_i:
6639         \skip_vertical:n { #1 }
6640         \egroup
6641     }
6642 }
6643 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6644 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6645 \cs_new:Npn \@@_MidRule_i:
6646 {
6647     \noalign \bgroup
6648     \peek_meaning:NTF [
6649         { \@@_MidRule_ii: }
6650         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6651     ]
6652 }
6653 \NewDocumentCommand \@@_MidRule_ii: { o }
6654 {
6655     \skip_vertical:N \aboverulesep
6656     \@@_create_row_node_i:
6657     \tl_gput_right:Ne \g_@@_rules_tl
6658     {
6659         \@@_draw_hruler:n

```

```

6658     {
6659         position = \int_eval:n { \c@iRow + 1 } ,
6660         tikz =
6661         {
6662             line-width = #1 ,
6663             yshift = 0.25 \arrayrulewidth ,
6664             shorten-< = - 0.5 \arrayrulewidth
6665         } ,
6666         total-width = #1 ,
6667     }
6668 }
6669 \skip_vertical:n { \belowrulesep + #1 }
6670 \egroup
6671 }

```

General system for drawing rules

```

6672 \AtBeginDocument
6673 {
6674     \cs_new_protected:Npe \@@_draw_rules:
6675     {
6676         \c_@@_pgfortikzpicture_tl
6677         \@@_draw_rules_i:
6678         \c_@@_endpgfortikzpicture_tl
6679     }
6680 }
6681 \cs_new_protected:Npn \@@_draw_rules_i:
6682 {
6683     \bool_lazy_all:nT
6684     {
6685         { \clist_if_empty_p:N \l_@@_corners_clist }
6686         { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6687         { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6688         { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6689         { ! \l_@@_fix_vertex_bool }
6690     }
6691     {
6692         \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6693         \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6694     }
6695     \pgfrememberpicturepositiononpagetrue
6696     \pgf@relevantforpicturesizefalse
6697     \pgfsetrectcap
6698     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6699     \CT@arc@
6700     \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6701     \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6702     \g_@@_rules_tl
6703 }

```

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”.

```

6704 \keys_define:nn { nicematrix / rules-after }
6705 {
6706     position .int_set:N = \l_@@_position_int ,
6707     start .int_set:N = \l_@@_start_int ,
6708     start .initial:n = 1 ,
6709     end .int_set:N = \l_@@_end_int ,
6710     multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6711     dotted .code:n =

```

```

6712     \bool_set_true:N \l_@@_dotted_bool
6713     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6714     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6715     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]`.

```

6716     color .code:n =
6717         \@@_set_CTarc:n { #1 }
6718         \CT@arc@
6719         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6720     color .value_required:n = true ,
6721     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6722     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
    }
}

6723     total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6724     unknown .code:n =
6725         \@@_unknown_key:nn
6726             { nicematrix / rules-after }
6727             { Unknown-key-for-a-rule } ,
6728     tikz .value_required:n = true
6729 }

```

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.

```

6730 \AtBeginDocument
6731 {
6732     \IfPackageLoadedTF { tikz }
6733     {
6734         \keys_define:nn { nicematrix / rules-after }
6735         {
6736             tikz .code:n =
6737                 \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6738                 \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6739                 \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6740             dashed .meta:n =
6741                 {
6742                     tikz = nicematrix / dashed ,
6743                     total-width = \pgflinewidth
6744                 }
6745         }
6746     }
6747     {
6748         \keys_define:nn { nicematrix / rules-after }
6749         { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6750     }
6751 }

```

```

6752 \cs_new_protected:Npn \@@_set_multiplicity:n #1

```

```

6753 {
6754   \int_set:Nn \l_@@_multiplicity_int { #1 }
6755   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6756   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6757 }
6758 \AddToHook { package / tikz / after }
6759 {
6760   \tikzset
6761   {
6762     nicematrix / dashed / .style =
6763     {
6764       dash-pattern = on~4~pt~off~2pt ,
6765       line-cap = butt
6766     }
6767   }
6768   \cs_if_exist:cT { tikz@library@decorations@loaded }
6769   { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6770 }

```

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.

```

6771 \cs_new_protected:Npn \@@_draw_vrule:n #1
6772 {

```

The group is for the options.

```

6773 {
6774   \int_set_eq:NN \l_@@_end_int \c_iRow
6775   \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).

```

6776   \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
6777   { \socket_use:n { nicematrix / draw-vrule } }
6778 }
6779 }

```

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.

```

6780 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6781 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6782 {
6783   \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`.

```

6784   \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).

```

6785 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6786 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6787 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6788 \l_tmpa_tl
6789 {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6790 \@@_test_v:
6791 \bool_if:NTF \g_tmpa_bool
6792 {
6793 \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.

```

6794 { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6795 { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6796 }
6797 {
6798 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6799 {
6800 \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6801 \socket_use:n { nicematrix / vsegment }
6802 \int_zero:N \l_@@_segment_start_int
6803 }
6804 }
6805 }
6806 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6807 {
6808 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6809 \socket_use:n { nicematrix / vsegment }
6810 }
6811 \group_end:
6812 }

6813 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6814 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6815 {
6816 \group_begin:
6817 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6818 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6819 \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6820 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6821 \socket_use:n { nicematrix / vsegment }
6822 \group_end:
6823 }

```

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.

```

6824 \cs_new_protected:Npn \@@_test_v:
6825 {
6826 \bool_gset_true:N \g_tmpa_bool
6827 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6828 \bool_if:NT \g_tmpa_bool
6829 {
6830 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6831 { \@@_test_vline_in_block:nnnnn ##1 }
6832 \bool_if:NT \g_tmpa_bool
6833 {
6834 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq

```

```

6835         { \@@_test_vline_in_block:nnnnn ##1 }
6836     \bool_if:NT \g_tmpa_bool
6837     {
6838         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6839         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6840     }
6841 }
6842 }
6843 }
6844 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6845 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6846 {
6847     {
6848         \int_compare:nNnTF \l_@@_position_int > \c@jCol
6849         { \bool_set_false:N \l_tmpa_bool }
6850         {
6851             \@@_test_h:
6852             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6853         }
6854         \int_compare:nNnTF \l_@@_position_int = 1
6855         { \bool_gset_false:N \g_tmpa_bool }
6856         {
6857             \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6858             \@@_test_h:
6859         }
6860         \bool_gset:Nn \g_tmpa_bool
6861         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6862     }
6863     \bool_if:NT \g_tmpa_bool
6864     {
6865         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6866         \socket_use:n { nicematrix / vsegment }
6867         \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6868     }
6869 }
6870 \cs_new_protected:Npn \@@_test_in_corner_v:
6871 {
6872     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6873     {
6874         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6875         { \bool_set_false:N \g_tmpa_bool }
6876     }
6877     {
6878         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6879         {
6880             \int_compare:nNnTF \l_tmpb_tl = 1
6881             { \bool_set_false:N \g_tmpa_bool }
6882             {
6883                 \@@_if_in_corner:nT
6884                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6885                 { \bool_set_false:N \g_tmpa_bool }
6886             }
6887         }
6888     }
6889 }

```

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.

```
6890 \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`.

```
6891 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6892 {
6893   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6894   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6895   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6896   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6897   \pgfusepathqstroke
6898 }
6899 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6900 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6901 {
6902   \group_begin:
6903   \dim_add:Nn \l_@@_x_initial_dim
6904     {
6905       - 0.5 \l_@@_rule_width_after_dim
6906       +
6907       ( \arrayrulewidth * \l_@@_multiplicity_int
6908         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6909     }
6910   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6911   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6912   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6913   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6914   \bool_lazy_and:nnT
6915     { \cs_if_exist_p:N \CT@drsc@ }
6916     { ! \tl_if_blank_p:o \CT@drsc@ }
6917     {
6918       {
6919         \CT@drsc@
6920         \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6921         \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6922         \dim_set:Nn \l_@@_tmpd_dim
6923           {
6924             \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6925             * ( \l_@@_multiplicity_int - 1 )
6926           }
6927         \pgfpathrectanglecorners
6928           { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6929           { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6930         \pgfusepath { fill }
6931       }
6932     }
6933   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6934   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6935   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6936   {
6937     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6938     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6939     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6940   }
6941   \pgfusepathqstroke
6942   \group_end:
6943 }
6944 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
```

```

6945 {
6946   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6947   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6948   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6949   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6950   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6951   \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`

```

6952   \@@_draw_line:
6953 }

```

```

6954 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6955 {
6956   {

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6957   \tl_if_empty:NF \l_@@_rule_color_tl
6958   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6959   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6960   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6961   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6962   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6963   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6964   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6965   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6966   ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6967 }
6968 }

```

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).

```

6969 \cs_new_protected:Npn \@@_draw_key_vlines:
6970 {
6971   {
6972     \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6973     \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currying in the following code.

```

6974   \str_if_eq:eeTF \l_@@_vlines_clist { all }
6975   { \@@_lines_step_inline:n \c@jCol }
6976   { \clist_map_inline:Nn \l_@@_vlines_clist }
6977   {
6978     \int_set:Nn \l_@@_position_int { ##1 }
6979     \int_compare:nNnTF { ##1 } > { \g_@@_col_total_int + 1 }
6980     { \@@_error:n { Column~outside~the~array } }

```

```

6981         { \socket_use:n { nicematrix / draw-vrule } }
6982     }
6983 }
6984 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6985 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6986 {
6987     \int_step_inline:nnn
6988     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6989     {
6990         #1
6991         \bool_lazy_or:nnF \g_@@_delims_bool \l_@@_except_borders_bool
6992         { + 1 }
6993     }
6994 }

```

The horizontal rules

`\@@_draw_hrule:n` and the socket `draw-hrule` will appear in `\@@_draw_key_hlines:n` and in the token rules `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6995 \cs_new_protected:Npn \@@_draw_hrule:n #1
6996 {
6997     {
6998         \int_set_eq:NN \l_@@_end_int \c@jCol
6999         \keys_set_known:nn { nicematrix / rules-after } { #1 }
7000         \socket_use:nn { nicematrix / draw-hrule }
7001     }
7002 }

```

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.

```

7003 \socket_new:nn { nicematrix / draw-hrule } { 0 }
7004 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
7005 {
7006     \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`.

```

7007     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }

```

`\l_@@_y_initial_dim` will be the *y*-value of the rule to draw (used in all the plugs).

```

7008     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
7009     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7010     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
7011     \l_tmpb_tl
7012     {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

7013 \@@_test_h:
7014 \bool_if:NTF \g_tmpa_bool
7015 {
7016 \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.

```

7017 { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
7018 { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
7019 }
7020 {
7021 \int_compare:nNtT \l_@@_segment_start_int > \c_zero_int
7022 {
7023 \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

7024 \socket_use:n { nicematrix / hsegment }
7025 \int_zero:N \l_@@_segment_start_int
7026 }
7027 }
7028 }
7029 \int_compare:nNtT \l_@@_segment_start_int > \c_zero_int
7030 {
7031 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7032 \socket_use:n { nicematrix / hsegment }
7033 }
7034 \group_end:
7035 }

7036 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
7037 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
7038 {
7039 \group_begin:
7040 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
7041 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7042 \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
7043 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7044 \socket_use:n { nicematrix / hsegment }
7045 \group_end:
7046 }

```

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.

```

7047 \cs_new_protected:Npn \@@_test_h:
7048 {
7049 \bool_gset_true:N \g_tmpa_bool
7050 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
7051 \bool_if:NT \g_tmpa_bool
7052 {
7053 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7054 { \@@_test_hline_in_block:nnnnn ##1 }
7055 \bool_if:NT \g_tmpa_bool
7056 {
7057 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
7058 { \@@_test_hline_in_block:nnnnn ##1 }
7059 \bool_if:NT \g_tmpa_bool
7060 {
7061 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
7062 { \@@_test_hline_in_stroken_block:nnnn ##1 }

```

```

7063     }
7064   }
7065 }
7066 }

7067 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
7068 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
7069 {
7070   {
7071     \int_compare:nNnTF \l_@@_position_int > \c@iRow
7072     { \bool_set_false:N \l_tmpa_bool }
7073     {
7074       \@@_test_v:
7075       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
7076     }
7077     \int_compare:nNnTF \l_@@_position_int = 1
7078     { \bool_gset_false:N \g_tmpa_bool }
7079     {
7080       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
7081       \@@_test_v:
7082     }
7083     \bool_gset:Nn \g_tmpa_bool
7084     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
7085   }
7086   \bool_if:NT \g_tmpa_bool
7087   {
7088     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
7089     \socket_use:n { nicematrix / hsegment }
7090     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
7091   }
7092 }

7093 \cs_new_protected:Npn \@@_test_in_corner_h:
7094 {
7095   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
7096   {
7097     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7098     { \bool_set_false:N \g_tmpa_bool }
7099   }
7100   {
7101     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
7102     {
7103       \int_compare:nNnTF \l_tmpa_tl = 1
7104       { \bool_set_false:N \g_tmpa_bool }
7105       {
7106         \@@_if_in_corner:nT
7107         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7108         { \bool_set_false:N \g_tmpa_bool }
7109       }
7110     }
7111   }
7112 }

```

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.

```
7113 \socket_new:nn { nicematrix / hsegment } { 0 }
```

In the following plug, the y -value for the line has been computed previously in $\backslash l_@@_y_initial_dim$.

```
7114 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
7115 {
7116   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7117   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7118   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7119   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7120   \pgfusepathqstroke
7121 }

7122 \socket_assign_plug:nn { nicematrix / hsegment } { standard }

7123 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
7124 {
7125   \group_begin:
7126   \dim_add:Nn \l_@@_y_initial_dim
7127   {
7128     - 0.5 \l_@@_rule_width_after_dim
7129     +
7130     ( \arrayrulewidth * \l_@@_multiplicity_int
7131       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
7132   }
7133   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7134   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7135   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7136   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7137   \bool_lazy_and:nnT
7138   { \cs_if_exist_p:N \CT@drsc@ }
7139   { ! \tl_if_blank_p:o \CT@drsc@ }
7140   {
7141     {
7142       \CT@drsc@
7143       \dim_set:Nn \l_@@_tmpd_dim
7144       {
7145         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
7146         * ( \l_@@_multiplicity_int - 1 )
7147       }
7148       \pgfpathrectanglecorners
7149       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7150       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
7151       \pgfusepathqfill
7152     }
7153   }
7154   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7155   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7156   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
7157   {
7158     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
7159     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7160     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7161   }
7162   \pgfusepathqstroke
7163   \group_end:
7164 }
```

Now, the case of a dotted line.

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

7165 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
7166 {
7167   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
7168   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
7169   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7170   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7171   \int_compare:nNnT \l_@@_segment_start_int = 1
7172   {
7173     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
7174     \bool_if:NF \g_@@_delims_bool
7175     { \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.

```

7176   \tl_if_eq:NnF \g_@@_left_delim_tl (
7177     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
7178   )
7179   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7180   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7181   \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
7182   {
7183     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
7184     \bool_if:NF \g_@@_delims_bool
7185     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
7186     \tl_if_eq:NnF \g_@@_right_delim_tl )
7187     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
7188   }

```

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`

```

7189   \@@_draw_line:
7190 }

```

```

7191 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
7192 {
7193   {

```

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).

```

7194 \tl_if_empty:NF \l_@@_rule_color_tl
7195 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
7196 \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7197 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7198 \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
7199 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7200 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7201 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
7202 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
7203 -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
7204 }
7205 }

```

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).

```

7206 \cs_new_protected:Npn \@@_draw_key_hlines:
7207 {
7208 {
7209 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
7210 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a curriification in the following code.

```

7211 \str_if_eq:eeTF \l_@@_hlines_clist { all }
7212 { \@@_lines_step_inline:n \c@iRow }
7213 { \clist_map_inline:Nn \l_@@_hlines_clist }
7214 {
7215 \int_set:Nn \l_@@_position_int { ##1 }
7216 \int_compare:nNnTF { ##1 } > { \g_@@_row_total_int + 1 }
7217 { \@@_error:n { Row-outside-the~array } }
7218 { \socket_use:n { nicematrix / draw-hrule } }
7219 }
7220 }
7221 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

7222 \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.

```

7223 \cs_set:Npn \@@_Hline_i:n #1
7224 {
7225 \peek_remove_spaces:n
7226 {
7227 \peek_meaning:NTF \Hline
7228 { \@@_Hline_ii:nn { #1 + 1 } }
7229 { \@@_Hline_iii:n { #1 } }
7230 }
7231 }

7232 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

7233 \cs_set:Npn \@@_Hline_iii:n #1
7234 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

7235 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
7236 {
7237 \@@_compute_rule_width:n { multiplicity = #1 , #2 }
7238 \skip_vertical:N \l_@@_rule_width_before_dim
7239 \tl_gput_right:Ne \g_@@_rules_tl
7240 {

```

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:

```
7241 {
7242   \int_compare:nNt { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
7243   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
7244   \dim_set:Nn \l_@@_rule_width_after_dim
7245     { \dim_use:N \l_@@_rule_width_before_dim }
7246   \@@_draw_hrule:n { #2 }
7247 }
7248 }
7249 \egroup
7250 }
```

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`).

```
7251 \cs_new_protected:Npn \@@_custom_line:n #1
7252 {
7253   \str_clear:N \l_@@_letter_str
7254   \str_clear:N \l_@@_command_str
7255   \str_clear:N \l_@@_ccommand_str
7256   \tl_clear_new:N \l_@@_other_keys_tl
7257   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
7258   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
7259   {
7260     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
```

We delete the last character which is a space.

```
7261   \str_set:Ne \l_@@_command_str
7262     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
7263 }
7264 \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
7265 {
7266   \str_set:Ne \l_@@_ccommand_str
7267     { \str_tail:N \l_@@_ccommand_str }
7268   \str_set:Ne \l_@@_ccommand_str
7269     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
7270 }
```

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.

```
7271 \bool_lazy_all:nTF
7272 {
7273   { \str_if_empty_p:N \l_@@_letter_str }
7274   { \str_if_empty_p:N \l_@@_command_str }
7275   { \str_if_empty_p:N \l_@@_ccommand_str }
7276 }
7277 { \@@_error:n { No~letter~and~no~command } }
```

```

7278     { \l_@@_custom_line_i:o \l_@@_other_keys_tl }
7279   }

7280 \keys_define:nn { nicematrix / custom-line }
7281 {
7282   letter .str_set:N = \l_@@_letter_str ,
7283   letter .value_required:n = true ,
7284   command .str_set:N = \l_@@_command_str ,
7285   command .value_required:n = true ,
7286   ccommand .str_set:N = \l_@@_ccommand_str ,
7287   ccommand .value_required:n = true
7288 }

7289 \cs_new_protected:Npn \l_@@_custom_line_i:n #1
7290 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7291   \bool_set_false:N \l_@@_tikz_rule_bool
7292   \bool_set_false:N \l_@@_dotted_rule_bool
7293   \bool_set_false:N \l_@@_color_bool

7294   \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7295   \bool_if:NT \l_@@_tikz_rule_bool
7296   {
7297     \IfPackageLoadedF { tikz }
7298     { \@@_error:n { tikz-in-custom-line-without-tikz } }
7299     \bool_if:NT \l_@@_color_bool
7300     { \@@_error:n { color-in-custom-line-with-tikz } }
7301   }
7302   \bool_if:NT \l_@@_dotted_rule_bool
7303   {
7304     \int_compare:nNnT \l_@@_multiplicity_int > 1
7305     { \@@_error:n { key-multiplicity-with-dotted } }
7306   }
7307   \str_if_empty:NF \l_@@_letter_str
7308   {
7309     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7310     { \@@_error:n { Several~letters } }
7311     {
7312       \tl_if_in:NoTF
7313       \c_@@_forbidden_letters_str
7314       \l_@@_letter_str
7315       { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
7316     }

```

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.

```

7317       \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7318       { \@@_v_custom_line:nn { #1 } }
7319     }
7320   }
7321 }
7322 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7323 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7324 }
7325 \cs_generate_variant:Nn \l_@@_custom_line_i:n { o }

7326 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbFVX|()[]!@<> }
7327 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbFVX|()[]!@<> }

```

The previous command `\l_@@_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.

```

7328 \keys_define:nn { nicematrix / custom-line-bis }
7329 {
7330   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7331   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7332   color .value_required:n = true ,
7333   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7334   tikz .value_required:n = true ,
7335   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7336   dotted .value_forbidden:n = true ,
7337   dashed .meta:n =
7338     {
7339       tikz = nicematrix / dashed ,
7340       total-width = \pgflinewidth
7341     } ,
7342   total-width .code:n = { } ,
7343   total-width .value_required:n = true ,
7344   sep-color .code:n = { } ,
7345   sep-color .value_required:n = true ,
7346   unknown .code:n =
7347     \@@_unknown_key:nn
7348       { nicematrix / custom-line-bis }
7349       { Unknown-key-for~custom-line }
7350 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7351 \bool_new:N \l_@@_dotted_rule_bool
7352 \bool_new:N \l_@@_tikz_rule_bool
7353 \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).

```

7354 \keys_define:nn { nicematrix / custom-line-width }
7355 {
7356   multiplicity .int_set:N = \l_@@_tmpc_int ,
7357   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7358   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7359     \bool_set_true:N \l_@@_total_width_bool ,
7360   total-width .value_required:n = true ,
7361   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7362 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the 'h' in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_draw_hrule:n` (which is in the internal `\CodeAfter`).

```

7363 \cs_new_protected:Npn \@@_h_custom_line:n #1
7364 {

```

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`).

```

7365   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7366   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7367 }

```

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`).

```

7368 \cs_new_protected:Npn \@@_c_custom_line:n #1
7369 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7370 \exp_args:Nc \DeclareExpandableDocumentCommand
7371 { nicematrix - \l_@@_ccommand_str }
7372 { 0 { } m }
7373 {
7374   \noalign
7375   {
7376     \@@_compute_rule_width:n { #1 , ##1 }
7377     \skip_vertical:n \l_@@_rule_width_before_dim
7378     \clist_map_inline:nn
7379       { ##2 }
7380       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7381   }
7382 }
7383 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7384 }

```

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*.

```

7385 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7386 {
7387   \tl_if_in:nnTF { #2 } { - }
7388   { \@@_cut_on_hyphen:w #2 \q_stop }
7389   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7390   \tl_gput_right:Ne \g_@@_rules_tl
7391   {
7392     \@@_draw_hrule:n
7393     {
7394       #1 ,
7395       start = \l_tmpa_tl ,
7396       end = \l_tmpb_tl ,
7397       position = \int_eval:n { \c@iRow + 1 } ,
7398       total-width = \dim_use:N \l_@@_rule_width_before_dim
7399     }
7400   }
7401 }

7402 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7403 {
7404   \bool_set_false:N \l_@@_tikz_rule_bool
7405   \bool_set_false:N \l_@@_total_width_bool
7406   \bool_set_false:N \l_@@_dotted_rule_bool
7407   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7408   \bool_if:NF \l_@@_total_width_bool
7409   {
7410     \bool_if:NTF \l_@@_dotted_rule_bool
7411     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7412     {
7413       \bool_if:NF \l_@@_tikz_rule_bool
7414       {
7415         \dim_set:Nn \l_@@_rule_width_before_dim
7416         {
7417           \arrayrulewidth * \l_@@_tmpc_int
7418           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7419         }
7420       }
7421     }
7422   }
7423 }

```

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

```

7424 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7425 {

```

```

7426 \str_if_eq:nnTF { #2 } { [ ]
7427 { \@@_v_custom_line_i:nw { #1 } [ ]
7428 { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7429 }
7430 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7431 { \@@_v_custom_line:nn { #1 , #2 } }
7432 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7433 {
7434 \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7435 \tl_put_right:Ne \l_@@_array_preamble_tl
7436 {
7437 \exp_not:N !
7438 { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7439 }
7440 \tl_gput_right:Ne \g_@@_rules_tl
7441 {

```

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.

```

7442 {
7443 \dim_set:Nn \l_@@_rule_width_after_dim
7444 { \dim_use:N \l_@@_rule_width_before_dim }

```

Here, the `\int_eval:n` is mandatory because we expand an instruction written on `\g_@@_rules_tl` (which will be executed later).

```

7445 \int_set:Nn \l_@@_position_int
7446 { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
7447 \@@_draw_vrule:n { #2 }
7448 }
7449 }
7450 \@@_rec_preamble:n #1
7451 }

7452 \@@_custom_line:n
7453 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

```

The key default-line

```

7454 \keys_define:nn { nicematrix / default-line }
7455 {
7456 multiplicity .int_set:N = \l_@@_multiplicity_int ,
7457 color .code:n = \bool_set_true:N \l_@@_color_bool ,
7458 color .value_required:n = true ,
7459 dashed .code:n = \@@_error:n { dashed-for-default-line~without~TikZ } ,
7460 dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7461 dotted .value_forbidden:n = true ,
7462 total-width .code:n = { } ,
7463 total-width .value_required:n = true ,
7464 sep-color .code:n = { } ,
7465 sep-color .value_required:n = true ,
7466 unknown .code:n =
7467 \@@_unknown_key:nn
7468 { nicematrix / default-line }
7469 { Unknown-key~for~default-line }

```

```

7470 }
7471 \AddToHook { package / tikz / after }
7472 {
7473   \keys_define:nn { nicematrix / default-line }
7474   {
7475     tikz .code:n =
7476       \bool_set_true:N \l_@@_tikz_rule_bool
7477       \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7478     tikz .value_required:n = true ,
7479     dashed .meta:n =
7480     {
7481       tikz = nicematrix / dashed ,
7482       total-width = \pgflinewidth
7483     }
7484   }
7485   \@@_custom_line:n
7486   {
7487     letter = ; ,
7488     command = \hdashedline ,
7489     ccommand = \cdashedline ,
7490     tikz = nicematrix / dashed ,
7491     total-width = \pgflinewidth
7492   }
7493 }
7494 \cs_new_protected:Npn \@@_default_line:n #1
7495 {
7496   \bool_set_false:N \l_@@_tikz_rule_bool
7497   \bool_set_false:N \l_@@_dotted_rule_bool
7498   \bool_set_false:N \l_@@_color_bool
7499   \keys_set:nn { nicematrix / default-line } { #1 }
7500   \bool_if:NT \l_@@_tikz_rule_bool
7501   {
7502     \IfPackageLoadedF { tikz }
7503     { \@@_error:n { tikz~in~custom~line~without~tikz } }
7504     \bool_if:NT \l_@@_color_bool
7505     { \@@_error:n { color~in~custom~line~with~tikz } }
7506   }
7507   \bool_if:NT \l_@@_dotted_rule_bool
7508   {
7509     \int_compare:nNnT \l_@@_multiplicity_int > 1
7510     { \@@_error:n { key~multiplicity~with~dotted } }
7511   }
7512   \bool_if:NTF \l_@@_tikz_rule_bool
7513   {
7514     \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7515     \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7516   }
7517   {
7518     \bool_if:NTF \l_@@_dotted_rule_bool
7519     {
7520       \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7521       \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7522     }
7523     {
7524       \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7525       \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7526     }
7527   }
7528 }

```

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`.

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
{
  \int_compare:nNnT \l_tmpa_tl > { #1 }
  {
    \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
    {
      \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
      {
        \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
        { \bool_gset_false:N \g_tmpa_bool }
      }
    }
  }
}
```

We will use a version a little more efficient.

```
7529 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7530 {
7531   \if_int_compare:w \l_tmpa_tl > #1
7532     \if_int_compare:w \l_tmpa_tl > #3 \else:
7533       \if_int_compare:w \l_tmpb_tl < #2 \else:
7534         \if_int_compare:w \l_tmpb_tl > #4 \else:
7535           \bool_gset_false:N \g_tmpa_bool
7536         \fi:
7537       \fi:
7538     \fi:
7539   \fi:
7540 }
```

The same for vertical rules.

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
{
  \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
  {
    \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
    {
      \int_compare:nNnT \l_tmpb_tl > { #2 }
      {
        \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
        { \bool_gset_false:N \g_tmpa_bool }
      }
    }
  }
}
```

We will use a version a little more efficient.

```
7541 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7542 {
7543   \if_int_compare:w \l_tmpa_tl < #1 \else:
7544     \if_int_compare:w \l_tmpa_tl > #3 \else:
7545       \if_int_compare:w \l_tmpb_tl > #2
7546         \if_int_compare:w \l_tmpb_tl > #4 \else:
7547           \bool_gset_false:N \g_tmpa_bool
7548         \fi:
7549       \fi:
7550     \fi:
7551   \fi:
7552 }
```

```

7550     \fi:
7551   \fi:
7552 }

```

Now, for the stroken blocks.

```

7553 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7554 {
7555   % \int_compare:nNnT \l_tmpb_tl < { #2 - 1 }
7556   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7557   {
7558     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7559     {
7560       \int_compare:nNnTF \l_tmpa_tl = { #1 }
7561       { \bool_gset_false:N \g_tmpa_bool }
7562       {
7563         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7564         { \bool_gset_false:N \g_tmpa_bool }
7565       }
7566     }
7567   }
7568 }

7569 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7570 {
7571   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7572   {
7573     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7574     {
7575       \int_compare:nNnTF \l_tmpb_tl = { #2 }
7576       { \bool_gset_false:N \g_tmpa_bool }
7577       {
7578         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7579         { \bool_gset_false:N \g_tmpa_bool }
7580       }
7581     }
7582   }
7583 }

```

23 The empty corners

```

7584 \cs_new_protected:Npn \@@_mark_blocks:
7585 {
7586   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7587   { \@@_mark_cells_of_block:nnnnn ##1 }
7588
7589   \cs_set_eq:NN \@@_mark_blocks: \prg_do_nothing:
7590 }

```

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.

```

7591 \cs_new_protected:Npn \@@_compute_corners:
7592 {
7593   \@@_mark_blocks:

```

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. However, for efficiency, the cells in the corners will also be marked directly in the main hash table of TeX.

```

7594   \clist_clear:N \l_@@_corners_cells_clist
7595   \clist_map_inline:Nn \l_@@_corners_clist

```

```

7596 {
7597   \str_case:nnF { ##1 }
7598   {
7599     { NW }
7600     { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7601     { NE }
7602     { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7603     { SW }
7604     { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7605     { SE }
7606     { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7607     { N }
7608     { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7609     { S }
7610     { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7611     { E }
7612     { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7613     { W }
7614     { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol }
7615   }
7616   { \@@_error:nn { bad~corner } { ##1 } }
7617 }

```

Even if the user has used the key corners, the list of cells in the corners may be empty. That's why we test against `\l_@@_corners_cells_clist` and not against `\l_@@_corners_clist`.

```

7618 \clist_if_empty:NF \l_@@_corners_cells_clist
7619 {

```

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.

```

7620   \tl_gput_right:Ne \g_@@_aux_tl
7621   {
7622     \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7623     { \l_@@_corners_cells_clist }
7624   }
7625 }
7626 }

```

```

7627 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7628 {
7629   \int_step_inline:nnn { #1 } { #3 }
7630   {
7631     \int_step_inline:nnn { #2 } { #4 }
7632     { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7633   }
7634 }

```

```

7635 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p , TF , T , F }
7636 {
7637   \cs_if_exist:cTF
7638   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7639   \prg_return_true:
7640   \prg_return_false:
7641 }

```

When we have to “mark” an empty cell, we will:

- put it in `\l_@@_corners_cells_clist`; that `clist` will be written on the aux file because you have to know the empty cells during the `\CodeBefore`;
- mark directly the empty cell by an entry in the main hash table of TeX (for efficiency).

```

7642 \cs_new_protected:Npn \@@_mark_empty_cell:nn #1 #2
7643 {
7644   \clist_put_right:Nx \l_@@_corners_cells_clist { #1 - #2 }
7645   \cs_set_nopar:cpn { @@ _ corner _ #1 - #2 } { }
7646 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner.

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.

```

7647 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7648 {

```

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.

```

7649   \int_zero_new:N \l_@@_last_empty_row_int
7650   \int_set:Nn \l_tmpa_int { #1 }
7651   \bool_set_false:N \l_tmpa_bool
7652   \bool_do_until:Nn \l_tmpa_bool
7653   {
7654     \bool_lazy_or:nnTF
7655     {
7656       \cs_if_exist_p:c
7657       {
7658         pgf @ sh @ ns @ \@@_env: -
7659         \int_use:N \l_tmpa_int - \int_eval:n { #2 }
7660       }
7661     }
7662     { \@@_if_in_block_p:nn \l_tmpa_int { #2 } }
7663     {
7664       \bool_set_true:N \l_tmpa_bool
7665       \int_compare:nNnTF \l_tmpa_int = { #1 }
7666       { \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int }
7667       { \int_set:Nn \l_@@_last_empty_row_int { \l_tmpa_int - #3 } }
7668     }
7669     {
7670       \int_compare:nNnTF \l_tmpa_int = { #5 }
7671       {
7672         \bool_set_true:N \l_tmpa_bool
7673         \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int
7674       }
7675       { \int_add:Nn \l_tmpa_int { #3 } }
7676     }
7677   }

```

Now, you determine the last empty cell in the row of number 1.

```

7678   \int_zero_new:N \l_@@_last_empty_column_int
7679   \int_set:Nn \l_tmpa_int { #2 }
7680   \bool_set_false:N \l_tmpa_bool
7681   \bool_do_until:Nn \l_tmpa_bool
7682   {
7683     \bool_lazy_or:nnTF
7684     {

```

```

7685     \cs_if_exist_p:c
7686     {
7687         pgf @ sh @ ns @ \@@_env: -
7688         \int_eval:n { #1 } - \int_use:N \l_tmpa_int
7689     }
7690 }
7691 { \@@_if_in_block_p:nn { #1 } \l_tmpa_int }
7692 {
7693     \bool_set_true:N \l_tmpa_bool
7694     \int_compare:nNnTF \l_tmpa_int = { #1 }
7695     { \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int }
7696     { \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 } }
7697 }
7698 {
7699     \int_compare:nNnTF \l_tmpa_int = { #6 }
7700     {
7701         \bool_set_true:N \l_tmpa_bool
7702         \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int
7703     }
7704     { \int_add:Nn \l_tmpa_int { #4 } }
7705 }
7706 }

```

Now, we loop over the rows.

```

7707     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7708     {

```

We treat the row number ##1 with a loop \bool_do_until:Nn.

```

7709         \int_set:Nn \l_tmpa_int { #2 }
7710         \bool_set_false:N \l_tmpa_bool
7711         \bool_do_until:Nn \l_tmpa_bool
7712         {
7713             \bool_lazy_or:nnTF
7714             {
7715                 \cs_if_exist_p:c
7716                 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7717             }
7718             { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7719             { \bool_set_true:N \l_tmpa_bool }
7720             {
7721                 \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7722                 \int_compare:nNnT \l_tmpa_int = \l_@@_last_empty_column_int
7723                 { \bool_set_true:N \l_tmpa_bool }
7724                 \int_add:Nn \l_tmpa_int { #4 }
7725             }
7726         }
7727         \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 }
7728     }
7729 }

```

For the “corners” N and S

```

7730 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7731 {
7732     \int_step_inline:nn { #1 }
7733     {
7734         \int_set:Nn \l_tmpa_int { #2 }
7735         \bool_set_false:N \l_tmpa_bool
7736         \bool_do_until:Nn \l_tmpa_bool
7737         {
7738             \bool_lazy_or:nnTF
7739             {
7740                 \cs_if_exist_p:c
7741                 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_tmpa_int - ##1 }
7742             }

```

```

7743         { \@@_if_in_block_p:nn \l_tmpa_int { ##1 } }
7744         { \bool_set_true:N \l_tmpa_bool }
7745         {
7746             \@@_mark_empty_cell:nn { \int_use:N \l_tmpa_int } { ##1 }
7747             \int_compare:nNnTF \l_tmpa_int = { #4 }
7748                 { \bool_set_true:N \l_tmpa_bool }
7749                 { \int_add:Nn \l_tmpa_int { #3 } }
7750         }
7751     }
7752 }
7753 }

```

For the “corners” E and W.

```

7754 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7755 {
7756     \int_step_inline:nn { #1 }
7757     {
7758         \int_set:Nn \l_tmpa_int { #2 }
7759         \bool_set_false:N \l_tmpa_bool
7760         \bool_do_until:Nn \l_tmpa_bool
7761             {
7762                 \bool_lazy_or:nnTF
7763                     {
7764                         \cs_if_exist_p:c
7765                             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7766                     }
7767                     { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7768                     { \bool_set_true:N \l_tmpa_bool }
7769                     {
7770                         \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7771                         \int_compare:nNnTF \l_tmpa_int = { #4 }
7772                             { \bool_set_true:N \l_tmpa_bool }
7773                             { \int_add:Nn \l_tmpa_int { #3 } }
7774                     }
7775             }
7776     }
7777 }

```

Of course, instead of the following lines, we could have used `\prg_new_conditional:Npnn`.

```

7778 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7779 \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 far less efficient:

```

\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...

```

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7780 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7781 \keys_define:nn { nicematrix / NiceMatrixBlock }
7782 {
7783     auto-columns-width .code:n =
7784     {
7785         \bool_set_true:N \l_@@_block_auto_columns_width_bool

```

```

7786     \dim_gzero_new:N \g_@@_max_cell_width_dim
7787     \bool_set_true:N \l_@@_auto_columns_width_bool
7788   }
7789 }

7790 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7791 {
7792   \int_gincr:N \g_@@_NiceMatrixBlock_int
7793   \dim_zero:N \l_@@_columns_width_dim
7794   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7795   \bool_if:NT \l_@@_block_auto_columns_width_bool
7796   {
7797     \cs_if_exist:cT
7798     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7799     {
7800       \dim_set:Nn \l_@@_columns_width_dim
7801       {
7802         \use:c
7803         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7804       }
7805     }
7806   }
7807 }

```

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`).

```

7808 {
7809   \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.

```

7810   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7811   {
7812     \bool_if:NT \l_@@_block_auto_columns_width_bool
7813     {
7814       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7815       \iow_shipout:Ne \@mainaux
7816       {
7817         \cs_gset:cpn
7818         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7819

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

7819         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7820       }
7821       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7822     }
7823   }
7824   \ignorespacesafterend
7825 }

```

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).

```

7826 \cs_new_protected:Npn \@@_create_extra_nodes:
7827 {

```

```

7828 \bool_if:nTF \l_@@_medium_nodes_bool
7829 {
7830   \bool_if:NTF \l_@@_no_cell_nodes_bool
7831   { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7832   {
7833     \bool_if:NTF \l_@@_large_nodes_bool
7834     \@@_create_medium_and_large_nodes:
7835     \@@_create_medium_nodes:
7836   }
7837 }
7838 {
7839   \bool_if:NT \l_@@_large_nodes_bool
7840   {
7841     \bool_if:NTF \l_@@_no_cell_nodes_bool
7842     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7843     \@@_create_large_nodes:
7844   }
7845 }
7846 }

```

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`.

```

7847 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7848 {
7849   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7850   {
7851     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7852     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7853     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7854     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7855   }
7856   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7857   {
7858     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7859     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7860     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7861     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7862   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7863 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7864 {
7865   \int_step_variable:nnNn
7866   \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.

```

7867     {
7868         \cs_if_exist:cT
7869         { 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.

```

7870     {
7871         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
7872         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7873         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7874         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7875         {
7876             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7877             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7878         }

```

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.

```

7879         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
7880         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7881         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7882         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7883         {
7884             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7885             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7886         }
7887     }
7888 }
7889 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7890 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7891 {
7892     \dim_compare:nNnT
7893     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7894     {
7895         \@@_qpoint:n { row - \@@_i: - base }
7896         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7897         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7898     }
7899 }
7900 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7901 {
7902     \dim_compare:nNnT
7903     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7904     {
7905         \@@_qpoint:n { col - \@@_j: }
7906         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7907         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7908     }
7909 }
7910 }

```

Here is the command \@@_create_medium_nodes:. When this command is used, the “medium nodes” are created.

```

7911 \cs_new_protected:Npn \@@_create_medium_nodes:
7912 {
7913     \pgfpicture
7914     \pgfrememberpicturepositiononpagetrue
7915     \pgf@relevantforpicturesizefalse
7916     \@@_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”.

```

7917     \tl_set:Nn \l_@@_suffix_tl { -medium }
7918     \@@_create_nodes:
7919     \endpgfpicture
7920 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁶. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7921 \cs_new_protected:Npn \@@_create_large_nodes:
7922 {
7923     \pgfpicture
7924     \pgfrememberpicturepositiononpagetrue
7925     \pgf@relevantforpicturesizefalse
7926     \@@_computations_for_medium_nodes:
7927     \@@_computations_for_large_nodes:
7928     \tl_set:Nn \l_@@_suffix_tl { - large }
7929     \@@_create_nodes:
7930     \endpgfpicture
7931 }
7932 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7933 {
7934     \pgfpicture
7935     \pgfrememberpicturepositiononpagetrue
7936     \pgf@relevantforpicturesizefalse
7937     \@@_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”.

```

7938     \tl_set:Nn \l_@@_suffix_tl { - medium }
7939     \@@_create_nodes:
7940     \@@_computations_for_large_nodes:
7941     \tl_set:Nn \l_@@_suffix_tl { - large }
7942     \@@_create_nodes:
7943     \endpgfpicture
7944 }

```

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.

```

7945 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7946 {
7947     \int_set:Nn \l_@@_first_row_int 1
7948     \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`.

```

7949     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7950     {
7951         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7952         {
7953             (
7954                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7955                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7956             )
7957             / 2
7958         }

```

¹⁶If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7959     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7960     { l_@@_row _ \@@_i: _min_dim }
7961   }
7962   \int_step_variable:nnN { \c@jCol - 1 } \@@_j:
7963   {
7964     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7965     {
7966       (
7967         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7968         \dim_use:c
7969         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7970       )
7971       / 2
7972     }
7973     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7974     { l_@@_column _ \@@_j: _ max _ dim }
7975   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7976   \dim_sub:cn
7977   { l_@@_column _ 1 _ min _ dim }
7978   \l_@@_left_margin_dim
7979   \dim_add:cn
7980   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7981   \l_@@_right_margin_dim
7982 }

```

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).

```

7983 \cs_new_protected:Npn \@@_create_nodes:
7984 {
7985   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7986   {
7987     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7988     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7989     \@@_pgf_rect_node:nnnnn
7990     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7991     { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
7992     { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
7993     { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
7994     { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
7995     \str_if_empty:NF \l_@@_name_str
7996     {
7997       \pgfnodealias
7998       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7999       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
8000     }
8001   }
8002 }
8003 \int_step_inline:nn \c@iRow
8004 {
8005   \pgfnodealias
8006   { \@@_env: - #1 - last \l_@@_suffix_tl }
8007   { \@@_env: - #1 - \int_use:N \c@jCol \l_@@_suffix_tl }
8008 }
8009 \int_step_inline:nn \c@jCol
8010 {

```

```

8011 \pgfnodealias
8012 { \@@_env: - last - ##1 \l_@@_suffix_tl }
8013 { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
8014 }
8015 \pgfnodealias
8016 { \@@_env: - last - last \l_@@_suffix_tl }
8017 { \@@_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 .

```

8018 \seq_map_pairwise_function:NNN
8019 \g_@@_multicolumn_cells_seq
8020 \g_@@_multicolumn_sizes_seq
8021 \@@_node_for_multicolumn:nn
8022 }

8023 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
8024 {
8025   \cs_set_nopar:Npn \@@_i: { #1 }
8026   \cs_set_nopar:Npn \@@_j: { #2 }
8027 }

```

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”).

```

8028 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
8029 {
8030   \@@_extract_coords_values: #1 \q_stop
8031   \@@_pgf_rect_node:nnnnn
8032   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
8033   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
8034   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
8035   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
8036   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
8037   \str_if_empty:NF \l_@@_name_str
8038   {
8039     \pgfnodealias
8040     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
8041     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
8042   }
8043 }

```

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).

```

8044 \keys_define:nn { nicematrix / BlockFirstPass }
8045 {
8046   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8047   \bool_set_true:N \l_@@_p_block_bool ,
8048   j .value_forbidden:n = true ,
8049   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8050   l .value_forbidden:n = true ,

```

```

8051   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8052   r .value_forbidden:n = true ,
8053   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8054   c .value_forbidden:n = true ,
8055   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8056   L .value_forbidden:n = true ,
8057   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8058   R .value_forbidden:n = true ,
8059   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8060   C .value_forbidden:n = true ,
8061   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8062   t .value_forbidden:n = true ,
8063   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8064   T .value_forbidden:n = true ,
8065   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8066   b .value_forbidden:n = true ,
8067   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8068   B .value_forbidden:n = true ,
8069   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8070   m .value_forbidden:n = true ,
8071   v-center .meta:n = m ,
8072   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8073   p .value_forbidden:n = true ,
8074   respect-arraystretch .code:n =
8075     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8076   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.

```

8077   color .code:n =
8078     \@@_color:n { #1 }
8079     \tl_set_rescan:Nnn
8080       \l_@@_draw_tl
8081       { \char_set_catcode_other:N ! }
8082       { #1 } ,
8083   color .value_required:n = true ,
8084 }

```

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.

```

8085 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

8086 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
8087 {

```

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).

```

8088   \tl_if_blank:nTF { #2 }
8089     { \@@_Block_ii:nnnnn 1 1 }
8090     {
8091       \tl_if_in:nnTF { #2 } { - }
8092       {
8093         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
8094         \@@_Block_i_czech:w \@@_Block_i:w
8095         #2 \q_stop
8096       }
8097       {
8098         \@@_error:nn { Bad~argument~for~Block } { #2 }
8099         \@@_Block_ii:nnnnn 1 1
8100       }
8101     }
8102   { #1 } { #3 } { #4 }

```

```

8103 \ignorespaces
8104 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

8105 \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`.

```

8106 {
8107   \char_set_catcode_active:N -
8108   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
8109 }

```

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.

```

8110 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
8111 {

```

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).

```

8112   \int_set:Nn \l_tmpa_int
8113   {
8114     \bool_lazy_or:nnTF
8115     { \tl_if_blank_p:n { #1 } }
8116     { \str_if_eq_p:ee { * } { #1 } }
8117     { 100 }
8118     { #1 }
8119   }
8120   \int_set:Nn \l_tmpb_int
8121   {
8122     \bool_lazy_or:nnTF
8123     { \tl_if_blank_p:n { #2 } }
8124     { \str_if_eq_p:ee { * } { #2 } }
8125     { 100 }
8126     { #2 }
8127   }

```

If the block is mono-column.

```

8128   \int_compare:nNnTF \l_tmpb_int = 1
8129   {
8130     \tl_if_empty:NTF \l_@@_hpos_cell_tl
8131     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
8132     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
8133   }
8134   { \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.

```

8135   \keys_set:known:n { nicematrix / BlockFirstPass } { #3 }
8136   \tl_set:Ne \l_tmpa_tl
8137   {
8138     { \int_use:N \c@iRow }
8139     { \int_use:N \c@jCol }
8140     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
8141     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
8142   }

```

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).

```
8143 \bool_set_false:N \l_tmpa_bool
8144 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
8145 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
8146 \bool_case:nF
8147 {
8148     \l_tmpa_bool { \@@_Block_vii:eenenn }
8149     \l_@@_p_block_bool { \@@_Block_vi:eenenn }
```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```
8150 \l_@@_X_bool { \@@_Block_v:eenenn }
8151 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eenenn }
8152 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
8153 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
8154 }
8155 { \@@_Block_v:eenenn }
8156 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
8157 }
```

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.

```
8158 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
8159 {
8160     \int_gincr:N \g_@@_block_box_int
8161     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8162     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
8163     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8164     {
8165         \tl_gput_right:Ne \g_@@_rules_tl
8166         {
8167             \@@_draw_diagbox:nnnnnn
8168             { \int_use:N \c@iRow }
8169             { \int_use:N \c@jCol }
8170             { \int_eval:n { \c@iRow + #1 - 1 } }
8171             { \int_eval:n { \c@jCol + #2 - 1 } }
8172             { \g_@@_row_style_tl \exp_not:n { ##1 } }
8173             { \g_@@_row_style_tl \exp_not:n { ##2 } }
8174         }
8175     }
8176     \box_gclear_new:c
8177     { 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!

```
8178 \hbox_gset:cn
8179 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
8180 {
```

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`).

```
8181 \tl_if_empty:NTF \l_@@_color_tl
8182 { \int_compare:nNnT { #2 } = 1 \set@color }
8183 { \@@_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`.

```
8184 \int_compare:nNnT { #1 } = 1
8185 {
8186 \int_if_zero:nTF \c@iRow
8187 {
```

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`.

```
$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$
```

```
8188 \cs_set_eq:NN \Block \@@_NullBlock:
8189 \l_@@_code_for_first_row_tl
8190 }
8191 {
8192 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8193 {
8194 \cs_set_eq:NN \Block \@@_NullBlock:
8195 \l_@@_code_for_last_row_tl
8196 }
8197 }
8198 \g_@@_row_style_tl
8199 }
```

The following command will be no-op when `respect-arraystretch` is in force.

```
8200 \@@_reset_arraystretch:
8201 \dim_zero:N \extrarowheight
```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```
8202 #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.).

```
8203 \@@_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}`.

```
8204 \bool_if:NTF \l_@@_tabular_bool
8205 {
8206   \bool_lazy_all:nTF
8207   {
8208     { \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.

```
8209     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
8210     { ! \g_@@_rotate_bool }
8211   }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
8212   {
8213     \use:e
8214     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
8215       \exp_not:N \begin { minipage }
8216       [ \str_lowercase:f \l_@@_vpos_block_str ]
8217       { \l_@@_col_width_dim }
8218       \str_case:on \l_@@_hpos_block_str
8219       { c \centering r \raggedleft l \raggedright }
8220     }
8221     #5
8222   \end { minipage }
8223 }
```

In the other cases, we use a `{tabular}`.

```
8224   {
8225     \use:e
8226     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
8227       \exp_not:N \begin { tabular }
8228       [ \str_lowercase:f \l_@@_vpos_block_str ]
8229       { @ { } \l_@@_hpos_block_str @ { } }
8230     }
8231     #5
8232   \end { tabular }
8233 }
8234 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
8235   {
8236     $ % $
8237     \bool_if:NT \l_@@_small_bool
8238     {
8239       \def \arraystretch { 0.47 }
8240       \dim_set:Nn \arraycolsep { 1.45 pt }
8241     }
8242     \use:e
8243     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

8244         \exp_not:N \begin { array }
8245         [ \str_lowercase:f \l_@@_vpos_block_str ]
8246         { @ { } \l_@@_hpos_block_str @ { } }
8247     }
8248     \@@_tuning_key_small:
8249     #5
8250     \end { array }
8251     $ % $
8252 }
8253 }
```

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).

```

8254 \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.

```

8255 \int_compare:nNnT { #2 } = 1
8256 {
8257     \dim_gset:Nn \g_@@_blocks_wd_dim
8258     {
8259         \dim_max:nn
8260         \g_@@_blocks_wd_dim
8261         {
8262             \box_wd:c
8263             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8264         }
8265     }
8266 }
```

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.

```

8267 \int_compare:nNnT { #1 } = 1
8268 {
8269     \bool_lazy_any:nT
8270     {
8271         { \str_if_empty_p:N \l_@@_vpos_block_str }
8272         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8273         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8274     }
8275     \@@_adjust_blocks_ht_dp:
8276 }
8277 \seq_gput_right:Ne \g_@@_blocks_seq
8278 {
8279     \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.

```

8280 {
8281     \exp_not:n { #3 } ,
8282     \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

8283 \bool_if:NT \g_@@_rotate_bool
8284 {
8285     \bool_if:NTF \g_@@_rotate_c_bool
8286     { m }
```

```

8287         {
8288             \int_compare:nNtT \c@iRow = \l_@@_last_row_int
8289             { T }
8290         }
8291     }
8292 }
8293 {
8294     \box_use_drop:c
8295     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8296 }
8297 }
8298 \bool_set_false:N \g_@@_rotate_c_bool
8299 }
8300 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
8301 {
8302     \dim_gset:Nn \g_@@_blocks_ht_dim
8303     {
8304         \dim_max:nn
8305         \g_@@_blocks_ht_dim
8306         {
8307             \box_ht:c
8308             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8309         }
8310     }
8311     \dim_gset:Nn \g_@@_blocks_dp_dim
8312     {
8313         \dim_max:nn
8314         \g_@@_blocks_dp_dim
8315         {
8316             \box_dp:c
8317             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8318         }
8319     }
8320 }
8321 \cs_new:Npn \@@_adjust_hpos_rotate:
8322 {
8323     \bool_if:NT \g_@@_rotate_bool
8324     {
8325         \str_set:Ne \l_@@_hpos_block_str
8326         {
8327             \bool_if:NTF \g_@@_rotate_c_bool
8328             c
8329             {
8330                 \str_case:onF \l_@@_vpos_block_str
8331                 { b l B l t r T r }
8332                 {
8333                     \int_compare:nNtTF \c@iRow = \l_@@_last_row_int
8334                     r
8335                     l
8336                 }
8337             }
8338         }
8339     }
8340 }
8341 \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*.

```

8342 \cs_new_protected:Npn \@@_rotate_box_of_block:
8343 {

```

```

8344 \box_grotate:cn
8345 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8346 { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8347 \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8348 {
8349   \vbox_gset_top:cn
8350   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8351   {
8352     \skip_vertical:n { 0.8 ex }
8353     \box_use:c
8354     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8355   }
8356 }
8357 \bool_if:NT \g_@@_rotate_c_bool
8358 {
8359   \hbox_gset:cn
8360   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8361   {
8362     \vbox_center:n
8363     {
8364       \box_use:c
8365       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8366     }
8367   }
8368 }
8369 }

```

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.

```

8370 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8371 {
8372   \seq_gput_right:Ne \g_@@_blocks_seq
8373   {
8374     \l_tmpa_tl
8375     { \exp_not:n { #3 } }
8376     {
8377       \bool_if:NTF \l_@@_tabular_bool
8378       {
8379         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8380 \@@_reset_arraystretch:
8381 \exp_not:n
8382 {
8383   \dim_zero:N \extrarowheight
8384   #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).

```

8385 \tag_if_active:T { \tag_stop:n { table } }
8386 \use:e
8387 {
8388   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8389   { @ { } \l_@@_hpos_block_str @ { } }
8390 }

```

```

8391             #5
8392         \end { tabular }
8393     }
8394 }
8395 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

8396     {
8397     {

```

The following will be no-op when respect-arraystretch is in force.

```

8398         \@@_reset_arraystretch:
8399         \exp_not:n
8400         {
8401             \dim_zero:N \extrarowheight
8402             #4
8403             $ % $
8404             \use:e
8405             {
8406                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8407                 { @ { } \l_@@_hpos_block_str @ { } }
8408             }
8409             \@@_tuning_key_small:
8410             #5
8411             \end { array }
8412             $ % $
8413         }
8414     }
8415 }
8416 }
8417 }
8418 }
8419 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

8420 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8421 {
8422     \seq_gput_right:Ne \g_@@_blocks_seq
8423     {
8424         \l_tmpa_tl
8425         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8426         { { \exp_not:n { #4 #5 } } }
8427     }
8428 }
8429 \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.

```

8430 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8431 {
8432     \seq_gput_right:Ne \g_@@_blocks_seq
8433     {
8434         \l_tmpa_tl
8435         { \exp_not:n { #3 } }
8436         { \exp_not:n { #4 #5 } }
8437     }
8438 }
8439 \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).

```

8440 \keys_define:nn { nicematrix / Block }
8441 {
8442   empty .code:n =
8443     \bool_set_true:N \l_@@_empty_block_bool
8444     \bool_set_true:N \l_@@_transparent_bool ,
8445   empty .value_forbidden:n = true ,
8446   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8447   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8448   tikz .code:n =
8449     \IfPackageLoadedTF { tikz }
8450     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8451     { \@@_error:n { tikz~key~without~tikz } } ,
8452   tikz .value_required:n = true ,
8453   fill .code:n =
8454     \tl_set_rescan:Nnn
8455     \l_@@_fill_tl
8456     { \char_set_catcode_other:N ! }
8457     { #1 } ,
8458   fill .value_required:n = true ,

```

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

```

8459   opacity .tl_set:N = \l_@@_opacity_tl ,
8460   opacity .value_required:n = true ,
8461   draw .code:n =
8462     \tl_set_rescan:Nnn
8463     \l_@@_draw_tl
8464     { \char_set_catcode_other:N ! }
8465     { #1 } ,
8466   draw .default:n = default ,
8467   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8468   rounded-corners .default:n = 4 pt ,
8469   color .code:n =
8470     \@@_color:n { #1 }
8471     \tl_set_rescan:Nnn
8472     \l_@@_draw_tl
8473     { \char_set_catcode_other:N ! }
8474     { #1 } ,
8475   borders .clist_set:N = \l_@@_borders_clist ,
8476   borders .default:n = all ,
8477   hvlines .meta:n = { vlines , hlines } ,
8478   vlines .bool_set:N = \l_@@_vlines_block_bool ,
8479   hlines .bool_set:N = \l_@@_hlines_block_bool ,
8480   rules/width .dim_set:N = \arrayrulewidth ,
8481   rules/color .tl_set:N = \l_@@_rules_color_tl ,
8482   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8483   line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8484   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8485     \bool_set_true:N \l_@@_p_block_bool ,
8486   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8487   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8488   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8489   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8490     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8491   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8492     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8493   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8494     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8495   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8496   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,

```

```

8497   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8498   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8499   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8500   m .value_forbidden:n = true ,
8501   v-center .meta:n = m ,
8502   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8503   p .value_forbidden:n = true ,
8504   name .str_set:N = \l_@@_block_name_str ,
8505   name .value_required:n = true ,
8506   respect-arraystretch .code:n =
8507     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8508   respect-arraystretch .value_forbidden:n = true ,
8509   transparent .bool_set:N = \l_@@_transparent_bool ,
8510   unknown .code:n =
8511     \@@_unknown_key:nn
8512     { nicematrix / Block }
8513     { Unknown-key-for-Block }
8514 }

```

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.

```

8515 \cs_new_protected:Npn \@@_draw_blocks:
8516 {
8517   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8518   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8519 }
8520 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8521 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8522   \int_zero:N \l_@@_last_row_int
8523   \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”).

```

8524   \int_compare:nNnTF { #3 } > { 98 }
8525   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8526   { \int_set:Nn \l_@@_last_row_int { #3 } }
8527   \int_compare:nNnTF { #4 } > { 98 }
8528   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8529   { \int_set:Nn \l_@@_last_col_int { #4 } }
8530   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8531   {
8532     \bool_lazy_and:nnTF
8533     \l_@@_preamble_bool
8534     {
8535       \int_compare_p:n
8536       { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8537     }
8538     { \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 } }
8539     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8540   }
8541   {
8542     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8543     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8544     {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8545     \use:e
8546     {
8547         \@@_Block_v:nnnnnn
8548         { #1 }
8549         { #2 }
8550         { \int_use:N \l_@@_last_row_int }
8551         { \int_use:N \l_@@_last_col_int }
8552     }
8553     { #5 }
8554     { #6 }
8555 }
8556 }
8557 }

```

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.

```

8558 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8559 {

```

The group is for the keys.

```

8560     \group_begin:
8561     \int_compare:nNtT { #1 } = { #3 }
8562     { \str_set:Nn \l_@@_vpos_block_str { t } }
8563     \keys_set:nn { nicematrix / Block } { #5 }
8564     \bool_if:NT \l_@@_empty_block_bool
8565     {
8566         \int_step_inline:nnn { #1 } { #3 }
8567         {
8568             \int_step_inline:nnn { #2 } { #4 }
8569             { \cs_undefine:c { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 } }
8570         }
8571     }

```

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`.

```

8572     \bool_if:NT \l_@@_amp_in_blocks_bool
8573     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }
8574     \bool_lazy_and:nnT
8575     \l_@@_vlines_block_bool
8576     { ! \l_@@_ampersand_bool }
8577     {
8578         \tl_gput_right:Ne \g_@@_rules_tl
8579         {
8580             \@@_vlines_block:nnnnnn
8581             { \dim_use:N \arrayrulewidth }
8582             { \l_@@_rules_color_tl }
8583             { #1 } { #2 } { #3 } { #4 }
8584         }
8585     }
8586     \bool_if:NT \l_@@_hlines_block_bool
8587     {
8588         \tl_gput_right:Ne \g_@@_rules_tl
8589         {
8590             \@@_hlines_block:nnnnnn
8591             { \dim_use:N \arrayrulewidth }
8592             { \l_@@_rules_color_tl }
8593             { #1 } { #2 } { #3 } { #4 }
8594         }
8595     }

```

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).

```

8596   \bool_if:NF \l_@@_transparent_bool
8597   {
8598     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8599     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8600   }

```

```

8601   \tl_if_empty:NF \l_@@_draw_tl
8602   {
8603     \tl_gput_right:Nn \g_@@_rules_tl
8604     {
8605       \@@_stroke_block:nnnnn

```

#5 are the options

```

8606       { #5 } { #1 } { #2 } { #3 } { #4 }
8607     }
8608     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8609     { { #1 } { #2 } { #3 } { #4 } }
8610   }
8611   \clist_if_empty:NF \l_@@_borders_clist
8612   {
8613     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8614     {
8615       \@@_stroke_borders_block:nnnnn
8616       { #5 } { #1 } { #2 } { #3 } { #4 }
8617     }
8618   }
8619   \tl_if_empty:NF \l_@@_fill_tl
8620   {
8621     \@@_add_opacity_to_fill:
8622     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8623     {
8624       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8625       { #1 - #2 }
8626       { #3 - #4 }
8627       { \dim_use:N \l_@@_rounded_corners_dim }
8628     }
8629   }
8630   \seq_if_empty:NF \l_@@_tikz_seq
8631   {
8632     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8633     {
8634       \@@_block_tikz:nnnnn
8635       { \seq_use:Nn \l_@@_tikz_seq { , } }
8636       { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8637     }
8638   }
8639   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8640   {
8641     \tl_gput_right:Ne \g_@@_rules_tl
8642     {
8643       \@@_draw_diagbox:nnnnnn
8644       { #1 } { #2 } { #3 } { #4 }
8645       { \exp_not:n { ##1 } }
8646       { \exp_not:n { ##2 } }
8647     }
8648   }

```

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.

```
8649 \pgfpicture
8650 \pgfrememberpicturepositiononpagetrue
8651 \pgf@relevantforpicturesizefalse
8652 \@@_qpoint:n { row - #1 }
8653 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8654 \@@_qpoint:n { col - #2 }
8655 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8656 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8657 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8658 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8659 \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.

```
8660 \@@_pgf_rect_node:nnnnn
8661 { \@@_env: - #1 - #2 - block }
8662 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8663 \str_if_empty:NF \l_@@_block_name_str
8664 {
8665   \pgfnodealias
8666   { \@@_env: - \l_@@_block_name_str }
8667   { \@@_env: - #1 - #2 - block }
8668   \str_if_empty:NF \l_@@_name_str
8669   {
8670     \pgfnodealias
8671     { \l_@@_name_str - \l_@@_block_name_str }
8672     { \@@_env: - #1 - #2 - block }
8673   }
8674 }
```

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.

```
8675 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8676 {
8677   \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.

```
8678 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8679 {
```

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.

```

8680         \cs_if_exist:cT
8681         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8682         {
8683             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8684             {
8685                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8686                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8687             }
8688         }
8689     }

```

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.

```

8690         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8691         {
8692             \@@_qpoint:n { col - #2 }
8693             \dim_set_eq:NN \l_tmpb_dim \pgf@x
8694         }
8695     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8696     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8697     {
8698         \cs_if_exist:cT
8699         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8700         {
8701             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8702             {
8703                 \pgfpointanchor
8704                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8705                 { east }
8706                 \dim_set:Nn \l_@@_tmpd_dim
8707                 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8708             }
8709         }
8710     }
8711     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8712     {
8713         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8714         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8715     }
8716     \@@_pgf_rect_node:nnnnn
8717     { \@@_env: - #1 - #2 - block - short }
8718     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8719 }

```

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.

```

8720     \bool_if:NT \l_@@_medium_nodes_bool
8721     {
8722         \@@_pgf_rect_node:nnn
8723         { \@@_env: - #1 - #2 - block - medium }
8724         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8725         {
8726             \pgfpointanchor
8727             { \@@_env:
8728                 - \int_use:N \l_@@_last_row_int
8729                 - \int_use:N \l_@@_last_col_int - medium
8730             }
8731             { south-east }
8732         }
8733     }
8734     \endpgfpicture
8735

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8736 \bool_if:NTF \l_@@_ampersand_bool
8737 {
8738   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8739   \int_zero_new:N \l_@@_split_int
8740   \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).

```

8741   \int_set:Nn \l_@@_first_row_int { #1 }
8742   \int_set:Nn \l_@@_first_col_int { #2 }
8743   \int_set:Nn \l_@@_last_row_int { #3 }
8744   \int_set:Nn \l_@@_last_col_int { #4 }

8745   \pgfpicture
8746   \pgfrememberpicturepositiononpagetrue
8747   \pgf@relevantforpicturesizefalse
8748   \@@_qpoint:n { row - #1 }
8749   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8750   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8751   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8752   \@@_qpoint:n { col - #2 }
8753   \dim_set_eq:NN \l_tmpa_dim \pgf@x
8754   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8755   \dim_set:Nn \l_tmpb_dim
8756     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8757   \bool_lazy_or:nnT
8758     \l_@@_vlines_block_bool
8759     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8760     {
8761       \int_step_inline:nn { \l_@@_split_int - 1 }
8762       {
8763         \pgfpathmoveto
8764         {
8765           \pgfpoint
8766             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8767             \l_@@_tmpc_dim
8768         }
8769         \pgfpathlineto
8770         {
8771           \pgfpoint
8772             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8773             \l_@@_tmpd_dim
8774         }
8775         \CT@arc@
8776         \pgfsetlinewidth { 1.1 \arrayrulewidth }
8777         \pgfsetrectcap
8778         \pgfusepathqstroke
8779       }
8780     }
8781   \cs_set_eq:NN \cellcolor \@@_subcellcolor
8782   \int_zero_new:N \l_@@_split_i_int

8783   \str_if_eq:eeTF \l_@@_vpos_block_str T
8784   {
8785     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8786     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8787   }
8788   {
8789     \str_if_eq:eeTF \l_@@_vpos_block_str B
8790     {
8791       \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8792       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8793     }

```

```

8794 {
8795     \bool_lazy_or:nnTF
8796     { \int_compare_p:nNn { #1 } = { #3 } }
8797     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8798     {
8799         \@@_qpoint:n { row - #1 - base }
8800         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8801     }
8802     {
8803         \str_if_eq:eeTF \l_@@_vpos_block_str b
8804         {
8805             \@@_qpoint:n { row - #3 - base }
8806             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8807         }
8808         {
8809             \@@_qpoint:n { #1 - #2 - block }
8810             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8811         }
8812     }
8813 }
8814 }
8815 \int_step_inline:nn \l_@@_split_int
8816 {
8817     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8818     \int_set:Nn \l_@@_split_i_int { ##1 }
8819     \dim_set:Nn \col@sep
8820     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8821     \pgftransformshift
8822     {
8823         \pgfpoint
8824         {
8825             \l_tmpa_dim + ##1 \l_tmpb_dim -
8826             \str_case:on \l_@@_hpos_block_str
8827             {
8828                 l { \l_tmpb_dim + \col@sep }
8829                 c { 0.5 \l_tmpb_dim }
8830                 r { \col@sep }
8831             }
8832         }
8833         { \l_@@_tmpc_dim }
8834     }
8835     \pgfset { inner~sep = \c_zero_dim }
8836     \pgfnode
8837     { rectangle }
8838     {
8839         \str_if_eq:eeTF T \l_@@_vpos_block_str
8840         {
8841             \str_case:on \l_@@_hpos_block_str
8842             {
8843                 l { north-west }
8844                 c { north }
8845                 r { north-east }
8846             }
8847         }
8848         {
8849             \str_if_eq:eeTF B \l_@@_vpos_block_str
8850             {
8851                 \str_case:on \l_@@_hpos_block_str
8852                 {
8853                     l { south-west }
8854                     c { south }
8855                     r { south-east }

```

```

8856     }
8857   }
8858   {
8859     \bool_lazy_any:nTF
8860     {
8861       { \int_compare_p:nNn { #1 } = { #3 } }
8862       { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8863       { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8864     }
8865     {
8866       \str_case:on \l_@@_hpos_block_str
8867       {
8868         l { base~west }
8869         c { base }
8870         r { base~east }
8871       }
8872     }
8873     {
8874       \str_case:on \l_@@_hpos_block_str
8875       {
8876         l { west }
8877         c { center }
8878         r { east }
8879       }
8880     }
8881   }
8882 }
8883 }
8884 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8885 \group_end:
8886 }
8887 \endpgfpicture
8888 }

```

Now the case where there is no ampersand & in the content of the block.

```

8889 {
8890   \bool_if:NTF \l_@@_p_block_bool
8891   {

```

When the final user has used the key p, we have to compute the width.

```

8892   \pgfpicture
8893   \pgfrememberpicturepositiononpagetrue
8894   \pgf@relevantforpicturesizefalse
8895   \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8896   {
8897     \@@_qpoint:n { col - #2 }
8898     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8899     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8900   }
8901   {
8902     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8903     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8904     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8905   }
8906   \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8907 \endpgfpicture
8908 \hbox_set:Nn \l_@@_cell_box
8909 {
8910   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8911     { \g_tmpb_dim }
8912     \str_case:on \l_@@_hpos_block_str
8913     { c \centering r \raggedleft l \raggedright j { } }
8914     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8915     #6

```

```

8916         \end { minipage }
8917     }
8918 }
8919 {
8920     \hbox_set:Nn \l_@@_cell_box
8921     {
8922         \set@color
8923         \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8924         #6
8925     }
8926 }
8927 \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.

```

8928 \pgfpicture
8929 \pgfrememberpicturepositiononpagetrue
8930 \pgf@relevantforpicturesizefalse
8931 \bool_lazy_any:nTF
8932 {
8933     { \str_if_empty_p:N \l_@@_vpos_block_str }
8934     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8935     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8936     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8937 }
8938 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8939     \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`.

```

8940     \bool_if:nT \g_@@_last_col_found_bool
8941     {
8942         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8943         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8944     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8945     \tl_set:Ne \l_tmpa_tl
8946     {
8947         \str_case:on \l_@@_vpos_block_str
8948         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8949             { } {
8950                 \str_case:on \l_@@_hpos_block_str
8951                 {
8952                     c { center }
8953                     l { west }
8954                     r { east }
8955                     j { center }
8956                 }
8957             }
8958         c {
8959             \str_case:on \l_@@_hpos_block_str
8960             {
8961                 c { center }
8962                 l { west }
8963                 r { east }
8964                 j { center }
8965             }
8966         }

```

```

8967     }
8968     T {
8969         \str_case:on \l_@@_hpos_block_str
8970         {
8971             c { north }
8972             l { north-west }
8973             r { north-east }
8974             j { north }
8975         }
8976
8977     }
8978     B {
8979         \str_case:on \l_@@_hpos_block_str
8980         {
8981             c { south }
8982             l { south-west }
8983             r { south-east }
8984             j { south }
8985         }
8986
8987     }
8988 }
8989
8990 \pgftransformshift
8991 {
8992     \pgfpointanchor
8993     {
8994         \@@_env: - #1 - #2 - block
8995         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8996     }
8997     \l_tmpa_tl
8998 }
8999 \pgfset { inner~sep = \c_zero_dim }
9000 \pgfnode
9001 { rectangle }
9002 \l_tmpa_tl
9003 { \box_use_drop:N \l_@@_cell_box } { } { }
9004 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

9005 {
9006     \pgfextracty \l_tmpa_dim
9007     {
9008         \@@_qpoint:n
9009         {
9010             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
9011             - base
9012         }
9013     }
9014     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

9015 \pgfpointanchor
9016 {
9017     \@@_env: - #1 - #2 - block
9018     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
9019 }
9020 {
9021     \str_case:on \l_@@_hpos_block_str
9022     {
9023         c { center }
9024         l { west }
9025         r { east }

```

```

9026         j { center }
9027     }
9028 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

9029     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
9030     \pgfset { inner~sep = \c_zero_dim }
9031     \pgfnode
9032     { rectangle }
9033     {
9034         \str_case:on \l_@@_hpos_block_str
9035         {
9036             c { base }
9037             l { base~west }
9038             r { base~east }
9039             j { base }
9040         }
9041     }
9042     { \box_use_drop:N \l_@@_cell_box } { } { }
9043 }
9044 \endpgfpicture
9045 }
9046 \group_end:
9047 }
9048 \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).

```

9049 \cs_new_protected:Npn \@@_add_opacity_to_fill:
9050 {
9051     \tl_if_empty:NF \l_@@_opacity_tl
9052     {
9053         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
9054             {
9055                 \tl_set:Nc \l_@@_fill_tl
9056                 {
9057                     [ opacity = \l_@@_opacity_tl ,
9058                     \tl_tail:o \l_@@_fill_tl
9059                 }
9060             }
9061             {
9062                 \tl_set:Nc \l_@@_fill_tl
9063                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
9064             }
9065         }
9066     }

```

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*.

```

9067 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
9068 {
9069     \group_begin:
9070     \tl_clear:N \l_@@_draw_tl
9071     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9072     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
9073     \tl_if_empty:NF \l_@@_draw_tl
9074     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

9075     \tl_if_eq:NnTF \l_@@_draw_tl { default }

```

```

9076         \CT@arc@
9077         { \@@_color:o \l_@@_draw_tl }
9078     }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

9079     \pgfsetcornersarced
9080     { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
9081     \int_compare:nNnF { #2 } > \c@iRow
9082     {
9083         \int_compare:nNnF { #3 } > \c@jCol
9084         {
9085             \@@_qpoint:n { row - #2 }
9086             \dim_set_eq:NN \l_tmpb_dim \pgf@y
9087             \@@_qpoint:n { col - #3 }
9088             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9089             \@@_qpoint:n
9090             { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
9091             \dim_set_eq:NN \l_tmpa_dim \pgf@y
9092             \@@_qpoint:n
9093             { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
9094             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
9095             \pgfpathrectanglecorners
9096             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
9097             { \pgfpoint \pgf@x \l_tmpa_dim }
9098             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
9099             \pgfusepathqstroke
9100             { \pgfusepath { stroke } }
9101         }
9102     }
9103     \group_end:
9104 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

9105 \keys_define:nn { nicematrix / BlockStroke }
9106 {
9107     color .tl_set:N = \l_@@_draw_tl ,
9108     draw .code:n =
9109         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
9110     draw .default:n = default ,
9111     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9112     rounded-corners .default:n = 4 pt ,
9113     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

9114     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9115 }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The second is the color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

9116 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9117 {
9118     \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).

```

9119     \dim_set:Nn \arrayrulewidth { #1 }
9120     \pgfsetlinewidth \arrayrulewidth
9121     \@@_set_CTarc:n { #2 }
9122     \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).

```

9123 \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9124 {
9125   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9126   ||
9127   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9128   ||
9129   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9130   ||
9131   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9132 }
9133 \bool_if:NT \l_@@_fix_vertex_bool
9134 {
9135   \int_compare:nNnT { #4 } > 1
9136   {
9137     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9138     {
9139       { 1 }
9140       { 1 }
9141       { \int_use:N \c@iRow }
9142       { \int_eval:n { #4 -1 } }
9143       { }
9144     }
9145   }
9146   \int_compare:nNnT { #6 } < \c@jCol
9147   {
9148     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9149     {
9150       { 1 }
9151       { \int_eval:n { #6 + 1 } }
9152       { \int_use:N \c@iRow }
9153       { \int_use:N \c@jCol }
9154       { }
9155     }
9156   }
9157 }
9158 \int_set:Nn \l_@@_start_int { #3 }
9159 \int_set:Nn \l_@@_end_int { #5 }
9160 \int_step_inline:nnn { #4 } { #6 + 1 }
9161 {
9162   \int_set:Nn \l_@@_position_int { ##1 }
9163   \socket_use:n { nicematrix / draw-vrule }
9164 }
9165 \group_end:
9166 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

9167 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9168 {
9169   \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).

```

9170 \dim_set:Nn \arrayrulewidth { #1 }
9171 \pgfsetlinewidth \arrayrulewidth
9172 \@@_set_CTarc:n { #2 }
9173 \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).

```

9174 \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9175 {
9176   \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9177   ||
9178   \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9179   ||
9180   \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9181   ||
9182   \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9183 }
9184 \bool_if:NT \l_@@_fix_vertex_bool
9185 {
9186   \int_compare:nNnT { #3 } > 1
9187   {
9188     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9189     {
9190       { 1 }
9191       { 1 }
9192       { \int_eval:n { #3 - 1 } }
9193       { \int_use:N \c@jCol }
9194       { }
9195     }
9196   }
9197   \int_compare:nNnT { #5 } < \c@iRow
9198   {
9199     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9200     {
9201       { \int_eval:n { #5 + 1 } }
9202       { 1 }
9203       { \int_use:N \c@iRow }
9204       { \int_use:N \c@jCol }
9205       { }
9206     }
9207   }
9208 }
9209 \int_set:Nn \l_@@_start_int { #4 }
9210 \int_set:Nn \l_@@_end_int { #6 }
9211 \int_step_inline:nnn { #3 } { #5 + 1 }
9212 {
9213   \int_set:Nn \l_@@_position_int { ##1 }
9214   \socket_use:n { nicematrix / draw-hrule }
9215 }
9216 \group_end:
9217 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

9218 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
9219 {
9220   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9221   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
9222
9223   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
9224   { \@@_error:n { borders~forbidden } }
9225   {
9226     \tl_clear_new:N \l_@@_borders_tikz_tl
9227     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
9228     \tl_set:Nn \l_@@_tmpc_tl { #2 }
9229     \tl_set:Nn \l_@@_tmpd_tl { #3 }
9230     \tl_set:Ne \l_@@_tmpa_tl { \int_eval:n { #4 + 1 } }

```

```

9231     \tl_set:Np \l_tmpb_tl { \int_eval:n { #5 + 1 } }
9232     \@@_stroke_borders_block_i:
9233 }
9234 }
9235 \AtBeginDocument
9236 {
9237     \cs_new_protected:Npe \@@_stroke_borders_block_i:
9238     {
9239         \c_@@_pgfortikzpicture_tl
9240         \@@_stroke_borders_block_ii:
9241         \c_@@_endpgfortikzpicture_tl
9242     }
9243 }
9244 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
9245 {
9246     \pgfrememberpicturepositiononpagetrue
9247     \pgf@relevantforpicturesizefalse
9248     \CT@arc@
9249     \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).

```

9250     \bool_set_false:N \l_tmpa_bool
9251     \clist_if_in:NnT \l_@@_borders_clist { right }
9252     {
9253         \@@_stroke_vertical:n \l_tmpb_tl
9254         \bool_set_true:N \l_tmpa_bool
9255     }
9256     \clist_if_in:NnT \l_@@_borders_clist { left }
9257     {
9258         \@@_stroke_vertical:n \l_@@_tmpd_tl
9259         \bool_set_true:N \l_tmpa_bool
9260     }
9261     \clist_if_in:NnT \l_@@_borders_clist { bottom }
9262     {
9263         \@@_stroke_horizontal:n \l_tmpa_tl
9264         \bool_set_true:N \l_tmpa_bool
9265     }
9266     \clist_if_in:NnT \l_@@_borders_clist { top }
9267     {
9268         \@@_stroke_horizontal:n \l_@@_tmpc_tl
9269         \bool_set_true:N \l_tmpa_bool
9270     }
9271     \bool_if:NF \l_tmpa_bool
9272     {
9273         \@@_stroke_vertical:n \l_tmpb_tl
9274         \@@_stroke_vertical:n \l_@@_tmpd_tl
9275         \@@_stroke_horizontal:n \l_tmpa_tl
9276         \@@_stroke_horizontal:n \l_@@_tmpc_tl
9277     }
9278 }
9279 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
9280 {
9281     tikz .code:n =
9282         \cs_if_exist:NTF \tikzpicture
9283         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
9284         { \@@_error:n { tikz~in~borders~without~tikz } } ,
9285     tikz .value_required:n = true ,
9286     dashed .meta:n = { tikz = dashed } ,
9287     top .code:n = ,
9288     bottom .code:n = ,
9289     left .code:n = ,

```

```

9290     right .code:n = ,
9291     all .code:n = ,
9292     unknown .code:n = \@@_error:n { bad~border }
9293 }

```

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).

```

9294 \cs_new_protected:Npn \@@_stroke_vertical:n #1
9295 {
9296   \@@_qpoint:n \l_@@_tmpc_tl
9297   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9298   \@@_qpoint:n \l_tmpa_tl
9299   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9300   \@@_qpoint:n { #1 }
9301   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9302   {
9303     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
9304     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
9305     \pgfusepathqstroke
9306   }
9307   {
9308     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9309     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
9310   }
9311 }

```

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).

```

9312 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
9313 {
9314   \@@_qpoint:n \l_@@_tmpd_tl
9315   \clist_if_in:NnTF \l_@@_borders_clist { left }
9316   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9317   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9318   \@@_qpoint:n \l_tmpb_tl
9319   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9320   \@@_qpoint:n { #1 }
9321   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9322   {
9323     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9324     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9325     \pgfusepathqstroke
9326   }
9327   {
9328     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9329     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9330   }
9331 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

9332 \keys_define:nn { nicematrix / BlockBorders }
9333 {
9334   borders .clist_set:N = \l_@@_borders_clist ,
9335   borders .default:n = all ,
9336   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9337   rounded-corners .default:n = 4 pt ,
9338   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9339   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9340 }

```

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.

```

9341 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9342 {
9343   \begin { tikzpicture }
9344   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

9345   \clist_map_inline:nn { #1 }
9346   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9347     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9348     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9349     (
9350       [
9351         xshift = \dim_use:N \l_@@_xoffset_dim ,
9352         yshift = - \dim_use:N \l_@@_yoffset_dim
9353       ]
9354       #2 -| #3
9355     )
9356     rectangle
9357     (
9358       [
9359         xshift = - \dim_use:N \l_@@_xoffset_dim ,
9360         yshift = \dim_use:N \l_@@_yoffset_dim
9361       ]
9362       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9363     ) ;
9364   }
9365   \end { tikzpicture }
9366 }
9367 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

9368 \keys_define:nn { nicematrix / SpecialOffset }
9369 {
9370   xoffset .dim_set:N = \l_@@_xoffset_dim ,
9371   yoffset .dim_set:N = \l_@@_yoffset_dim ,
9372   offset .meta:n = { xoffset = #1 , yoffset = #1 }
9373 }

```

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.

```

9374 \cs_new_protected:Npn \@@_NullBlock:
9375 { \@@_collect_options:n { \@@_NullBlock_i: } }
9376 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9377 { }

```

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.

```

9378 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9379 {
9380   \tl_gput_right:Ne \g_@@_pre_code_before_tl
9381   {

```

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).

```

9382 \@@_subcellcolor:nnnnnnn
9383 {
9384   \tl_if_blank:nTF { #1 }
9385     { { \exp_not:n { #2 } } }
9386     { [ #1 ] { \exp_not:n { #2 } } }
9387 }
9388 { \int_use:N \l_@@_first_row_int } % first row of the block
9389 { \int_use:N \l_@@_first_col_int } % first column of the block
9390 { \int_use:N \l_@@_last_row_int } % last row of the block
9391 { \int_use:N \l_@@_last_col_int } % last column of the block
9392 { \int_use:N \l_@@_split_int }
9393 { \int_use:N \l_@@_split_i_int }
9394 }
9395 \ignorespaces
9396 }
9397 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9398 {
9399   \@@_color_opacity: #1
9400   \pgfpicture
9401   \pgf@relevantforpicturesizefalse
9402   \@@_qpoint:n { col - #3 }
9403   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9404   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9405   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9406   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9407   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9408   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9409   \@@_qpoint:n { row - #2 }
9410   \pgfpathrectanglecorners
9411     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9412     { \pgfpoint \l_tmpb_dim \pgf@y }
9413   \pgfusepathqfill
9414   \endpgfpicture
9415 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

9416 \keys_define:nn { nicematrix / Auto }
9417 {
9418   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9419   columns-type .value_required:n = true ,
9420   l .meta:n = { columns-type = l } ,
9421   r .meta:n = { columns-type = r } ,
9422   c .meta:n = { columns-type = c } ,
9423   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9424   delimiters / color .value_required:n = true ,
9425   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9426   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9427   delimiters .value_required:n = true ,
9428   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9429   rounded-corners .default:n = 4 pt
9430 }
9431 \NewDocumentCommand \AutoNiceMatrixWithDelims
9432 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9433 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

```

```

9434 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9435 {

```

The group is for the protection of the keys.

```

9436 \group_begin:
9437 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9438 \use:e
9439 {
9440 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9441 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9442 [ \exp_not:o \l_tmpa_tl ]
9443 }
9444 \int_if_zero:nT \l_@@_first_row_int
9445 {
9446 \int_if_zero:nT \l_@@_first_col_int { & }
9447 \prg_replicate:nn { #4 - 1 } { & }
9448 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9449 }
9450 \prg_replicate:nn { #3 }
9451 {
9452 \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).

```

9453 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9454 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9455 }
9456 \int_compare:nNnT \l_@@_last_row_int > { -2 }
9457 {
9458 \int_if_zero:nT \l_@@_first_col_int { & }
9459 \prg_replicate:nn { #4 - 1 } { & }
9460 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9461 }
9462 \end { NiceArrayWithDelims }
9463 \group_end:
9464 }
9465 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9466 {
9467 \cs_set_protected:cpn { #1 AutoNiceMatrix }
9468 {
9469 \bool_gset_true:N \g_@@_delims_bool
9470 \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9471 \AutoNiceMatrixWithDelims { #2 } { #3 }
9472 }
9473 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

9474 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
9475 {
9476 \group_begin:
9477 \bool_gset_false:N \g_@@_delims_bool
9478 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9479 \group_end:
9480 }

```

28 The redefinition of the command \dotfill

```

9481 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9482 \cs_new_protected:Npn \@@_dotfill:
9483 {

```

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}`).

```

9484 \@@_old_dotfill:
9485 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9486 }

```

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`.

```

9487 \cs_new_protected:Npn \@@_dotfill_i:
9488 {
9489   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9490     { \@@_old_dotfill: }
9491 }

```

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.

```

9492 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9493 {
9494   \tl_gput_right:Ne \g_@@_rules_tl
9495   {
9496     \@@_draw_diagbox:nnnnnn
9497     { \int_use:N \c@iRow }
9498     { \int_use:N \c@jCol }
9499     { \int_use:N \c@iRow }
9500     { \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.

```

9501     { \g_@@_row_style_tl \exp_not:n { #1 } }
9502     { \g_@@_row_style_tl \exp_not:n { #2 } }
9503   }

```

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.

```

9504   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9505   {
9506     { \int_use:N \c@iRow }
9507     { \int_use:N \c@jCol }
9508     { \int_use:N \c@iRow }
9509     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9510     { }
9511   }
9512 }

```

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.

```

9513 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9514 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9515 \begin { pgfscope }
9516 \@@_qpoint:n { row - #1 }
9517 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9518 \@@_qpoint:n { col - #2 }
9519 \dim_set_eq:NN \l_tmpb_dim \pgf@x
9520 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9521 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9522 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9523 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9524 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9525 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9526 {

```

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.

```

9527 \CT@arc@
9528 \pgfsetroundcap
9529 \pgfusepathqstroke
9530 }
9531 \pgfset { inner~sep = 1 pt }
9532 \pgfscope
9533 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9534 \pgfnode { rectangle } { south~west }
9535 {
9536 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9537 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9538 \end { minipage }
9539 }
9540 { }
9541 { \pgfusepath { discard } } % mandatory
9542 \endpgfscope
9543 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9544 \pgfnode { rectangle } { north~east }
9545 {
9546 \begin { minipage } { 20 cm }
9547 \raggedleft
9548 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9549 \end { minipage }
9550 }
9551 { }
9552 { \pgfusepath { discard } } % mandatory
9553 \end { pgfscope }
9554 }

```

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. 92.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9555 \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 `\`.

```

9556 \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`.

```

9557 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9558 {
9559   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9560   \@@_CodeAfter_iv:n
9561 }

```

We catch the argument of the command `\end` (in `#1`).

```

9562 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9563 {

```

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.

```

9564   \str_if_eq:eeTF \@currenvir { #1 }
9565   { \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`.

```

9566   {
9567     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9568     \@@_CodeAfter_ii:n
9569   }
9570 }

```

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.

```

9571 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9572 {
9573   \pgfpicture
9574   \pgfrememberpicturepositiononpagetrue
9575   \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.

```

9576   \@@_qpoint:n { row - 1 }
9577   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - \arrayrulewidth }
9578   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9579   \dim_set:Nn \l_@@_y_final_dim { \pgf@y - \arrayrulewidth }

```

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).

```

9580   \bool_if:nTF { #3 }
9581   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9582   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9583   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9584   {
9585     \cs_if_exist:cT
9586     { pgf @ sh @ ns @ \@@_env: - #1 - #2 }

```

```

9587     {
9588         \pgfpointanchor
9589         { \@@_env: - ##1 - #2 }
9590         { \bool_if:nTF { #3 } { west } { east } }
9591         \dim_set:Nn \l_tmpa_dim
9592         {
9593             \bool_if:nTF { #3 }
9594             \dim_min:nn
9595             \dim_max:nn
9596             \l_tmpa_dim
9597             \pgf@x
9598         }
9599     }
9600 }

```

Now we can put the delimiter with a node of PGF.

```

9601 \pgfset { inner~sep = \c_zero_dim }
9602 \dim_zero:N \nulldelimiterspace
9603 \pgftransformshift
9604 {
9605     \pgfpoint
9606     \l_tmpa_dim
9607     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9608 }
9609 \pgfnode
9610 { rectangle }
9611 { \bool_if:nTF { #3 } { east } { west } }
9612 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9613     \nullfont
9614     $ % $
9615     \@@_color:o \l_@@_delimiters_color_tl
9616     \bool_if:nTF { #3 } { \left #1 } { \left . }
9617     \vcenter
9618     {
9619         \nullfont
9620         \hrule \@height
9621             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9622             \@depth \c_zero_dim
9623             \@width \c_zero_dim
9624     }
9625     \bool_if:nTF { #3 } { \right . } { \right #1 }
9626     $ % $
9627 }
9628 { }
9629 { }
9630 \endpgfpicture
9631 }

```

32 The command `\SubMatrix`

```

9632 \keys_define:nn { nicematrix / sub-matrix }
9633 {
9634     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9635     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9636     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9637     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9638     xshift .value_required:n = true ,
9639     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9640     delimiters / color .value_required:n = true ,

```

```

9641     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9642     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9643     hlines .default:n = all ,
9644     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9645     vlines .default:n = all ,
9646     hvlines .meta:n = { hlines, vlines } ,
9647     hvlines .value_forbidden:n = true
9648   }
9649 \keys_define:nn { nicematrix }
9650 {
9651   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9652   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9653   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9654   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9655 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9656 \keys_define:nn { nicematrix / SubMatrix }
9657 {
9658   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9659   delimiters / color .value_required:n = true ,
9660   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9661   hlines .default:n = all ,
9662   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9663   vlines .default:n = all ,
9664   hvlines .meta:n = { hlines, vlines } ,
9665   hvlines .value_forbidden:n = true ,
9666   name .code:n =
9667     \tl_if_empty:nTF { #1 }
9668     { \@@_error:n { Invalid-name } }
9669     {
9670       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9671       {
9672         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9673         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9674         {
9675           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9676           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9677         }
9678       }
9679       { \@@_error:n { Invalid-name } }
9680     } ,
9681   name .value_required:n = true ,
9682   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9683   rules .value_required:n = true ,
9684   code .tl_set:N = \l_@@_code_tl ,
9685   code .value_required:n = true ,
9686   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9687 }

9688 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9689 {
9690   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9691   {
9692     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9693     [
9694       delimiters / color = \l_@@_delimiters_color_tl ,
9695       hlines = \l_@@_submatrix_hlines_clist ,
9696       vlines = \l_@@_submatrix_vlines_clist ,
9697       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9698       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9699       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,

```

```

9700         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9701         #5
9702     ]
9703 }
9704 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9705 \ignorespaces
9706 }
9707 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9708 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9709 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
9710 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9711 {
9712     \seq_gput_right:Ne \g_@@_submatrix_seq
9713     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9714     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9715     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9716     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9717     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9718 }
9719 }

```

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).

```

9720 \NewDocumentCommand \@@_compute_i_j:nn
9721 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9722 { \@@_compute_i_j:nnnn #1 #2 }
9723 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9724 {
9725     \def \l_@@_first_i_tl { #1 }
9726     \def \l_@@_first_j_tl { #2 }
9727     \def \l_@@_last_i_tl { #3 }
9728     \def \l_@@_last_j_tl { #4 }
9729     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9730     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9731     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9732     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9733     \tl_if_eq:NnT \l_@@_last_i_tl { last }
9734     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9735     \tl_if_eq:NnT \l_@@_last_j_tl { last }
9736     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9737 }

```

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`.

```

9738 \AtBeginDocument
9739 {
9740   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
9741   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9742     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9743 }
9744 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9745 {
9746   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9747 \@@_compute_i_j:nn { #2 } { #3 }
9748 \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9749 { \def \arraystretch { 1 } }
9750 \bool_lazy_or:nnTF
9751 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9752 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9753 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9754 {
9755   \str_clear_new:N \l_@@_submatrix_name_str
9756   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9757   \pgfpicture
9758   \pgfrememberpicturepositiononpagetrue
9759   \pgf@relevantforpicturesizefalse
9760   \pgfset { inner~sep = \c_zero_dim }
9761   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9762   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9763 \bool_if:Ntf \l_@@_submatrix_slim_bool
9764 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9765 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9766 {
9767   \cs_if_exist:cT
9768   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9769   {
9770     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9771     \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9772     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9773   }
9774   \cs_if_exist:cT
9775   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9776   {
9777     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9778     \dim_compare:nNt \pgf@x > \l_@@_x_final_dim
9779     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9780   }
9781 }
9782 \dim_compare:nNtTF \l_@@_x_initial_dim = \c_max_dim
9783 { \@@_impossible_delimiter:n { left } }
9784 {
9785   \dim_compare:nNtTF \l_@@_x_final_dim = { - \c_max_dim }
9786   { \@@_impossible_delimiter:n { right } }
9787   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9788 }
9789 \endpgfpicture
9790 }
9791 \group_end:
9792 \ignorespaces
9793 }

```

The argument of the following command will be provided by curryfication.

```

9794 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9795 {
9796   \bool_if:NTF \l_@@_no_cell_nodes_bool
9797   { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9798   { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9799 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9800 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9801 {
9802   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9803   \dim_set:Nn \l_@@_y_initial_dim
9804   {
9805     \fp_to_dim:n
9806     {
9807       \pgf@y
9808       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9809       - \arrayrulewidth
9810     }
9811   }
9812   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9813   \dim_set:Nn \l_@@_y_final_dim
9814   {
9815     \fp_to_dim:n
9816     {
9817       \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch
9818       - \arrayrulewidth
9819     }
9820   }
9821   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9822   {
9823     \cs_if_exist:cT
9824     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9825     {
9826       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9827       \dim_set:Nn \l_@@_y_initial_dim
9828       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9829     }
9830     \cs_if_exist:cT
9831     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9832     {
9833       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9834       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9835       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9836     }
9837   }
9838   \dim_set:Nn \l_tmpa_dim
9839   {
9840     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9841     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9842   }
9843   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9844 \group_begin:
9845 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9846 \@@_set_CTarc:o \l_@@_rules_color_tl
9847 \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`.

```

9848 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9849 {
9850   \int_compare:nNt \l_@@_first_j_tl < { ##1 }
9851   {
9852     \int_compare:nNt
9853       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9854     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9855       \@@_qpoint:n { col - ##1 }
9856       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9857       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9858       \pgfusepathqstroke
9859     }
9860   }
9861 }

```

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.

```

9862 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9863 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9864 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9865 {
9866   \bool_lazy_and:nnTF
9867   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9868   {
9869     \int_compare_p:nNn
9870     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9871   {
9872     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9873     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9874     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9875     \pgfusepathqstroke
9876   }
9877   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9878 }

```

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.

```

9879 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9880 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9881 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9882 {
9883   \bool_lazy_and:nnTF
9884   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9885   {
9886     \int_compare_p:nNn
9887     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9888   {
9889     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9890 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

9891 \dim_set:Nn \l_tmpa_dim
9892 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9893 \str_case:nn { #1 }
9894 {
9895   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9896   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9897   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9898   }
9899   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

9900         \dim_set:Nn \l_tmpb_dim
9901         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9902         \str_case:nn { #2 }
9903         {
9904             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9905             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9906             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9907         }
9908         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9909         \pgfusepathqstroke
9910         \group_end:
9911     }
9912     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
9913 }

```

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).

```

9914     \str_if_empty:NF \l_@@_submatrix_name_str
9915     {
9916         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9917         \l_@@_x_initial_dim \l_@@_y_initial_dim
9918         \l_@@_x_final_dim \l_@@_y_final_dim
9919     }
9920     \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`.

```

9921     \begin { pgfscope }
9922     \pgftransformshift
9923     {
9924         \pgfpoint
9925         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9926         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9927     }
9928     \str_if_empty:NTF \l_@@_submatrix_name_str
9929     { \@@_node_left:nn #1 { } }
9930     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9931     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9932     \pgftransformshift
9933     {
9934         \pgfpoint
9935         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9936         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9937     }
9938     \str_if_empty:NTF \l_@@_submatrix_name_str
9939     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9940     {
9941         \@@_node_right:nnnn #2
9942         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9943     }

```

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`.

```

9944     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9945     \flag_clear_new:N \l_@@_code_flag
9946     \l_@@_code_tl
9947 }

```

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`.

```
9948 \cs_set_eq:NN \@@_old_pgfpntanchor: \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 `\@@_pgfpntanchor_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.

```
9949 \cs_new:Npn \@@_pgfpntanchor:n #1
9950 { \exp_args:Ne \@@_old_pgfpntanchor: { \@@_pgfpntanchor_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`).

```
9951 \cs_new:Npn \@@_pgfpntanchor_i:n #1
9952 { \@@_pgfpntanchor_ii:w #1 \tikz@pp@name \q_stop }

9953 \cs_new:Npn \@@_pgfpntanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9954 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9955 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9956 { \@@_pgfpntanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9957 { \@@_pgfpntanchor_ii:n { #1 } }
9958 }
```

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. `\@@_pgfpntanchor_i:n`).

```
9959 \cs_new:Npn \@@_pgfpntanchor_iv:w #1 \tikz@pp@name
9960 { \@@_pgfpntanchor_ii:n { #1 } }
```

With the command `\@@_pgfpntanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form i or of the form $i-j$ (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
9961 \cs_new:Npn \@@_pgfpntanchor_ii:n #1 { \@@_pgfpntanchor_i:w #1- \q_stop }
```

```
9962 \cs_new:Npn \@@_pgfpntanchor_i:w #1-#2 \q_stop
9963 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9964 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9965 { \@@_pgfpntanchor_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. `\@@_pgfpntanchor_ii:n`).

```
9966 { \@@_pgfpntanchor_iii:w { #1 } #2 }
9967 }
```

The following function is for the case when the name contains an hyphen.

```
9968 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9969 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9970   \@@_env:
9971   - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9972   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9973 }
```

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.

```
9974 \tl_const:Nn \c_@@_integers_alist_tl
9975 {
9976   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9977   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9978   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9979   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9980 }
```

```
9981 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9982 {
```

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`.

```
9983   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9984   {
9985     \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9986   \@@_env: -
9987   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9988     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9989     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9990   }
9991   {
9992     \str_if_eq:eeTF { #1 } { last }
9993     {
9994       \flag_raise:N \l_@@_code_flag
9995       \@@_env: -
9996       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9997         { \int_eval:n { \l_@@_last_i_tl + 1 } }
9998         { \int_eval:n { \l_@@_last_j_tl + 1 } }
9999     }
10000     { #1 }
10001   }
10002 }
```

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`).

```
10003 \cs_new_protected:Npn \@@_node_left:nn #1 #2
10004 {
10005   \pgfnode
10006   { rectangle }
```

```

10007 { east }
10008 {
10009   \nullfont
10010   $ % $
10011   \@@_color:o \l_@@_delimiters_color_tl
10012   \left #1
10013   \vcenter
10014   {
10015     \nullfont
10016     \hrule \@height \l_tmpa_dim
10017             \@depth \c_zero_dim
10018             \@width \c_zero_dim
10019   }
10020   \right .
10021   $ % $
10022 }
10023 { #2 }
10024 { }
10025 }

```

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.

```

10026 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
10027 {
10028   \pgfnode
10029   { rectangle }
10030   { west }
10031   {
10032     \nullfont
10033     $ % $
10034     \colorlet { current-color } { . }
10035     \@@_color:o \l_@@_delimiters_color_tl
10036     \left .
10037     \vcenter
10038     {
10039       \nullfont
10040       \hrule \@height \l_tmpa_dim
10041               \@depth \c_zero_dim
10042               \@width \c_zero_dim
10043     }
10044     \right #1
10045     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
10046     ^ { \color { current-color } \smash { #4 } }
10047     $ % $
10048   }
10049   { #2 }
10050   { }
10051 }

```

33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

10052 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
10053 {
10054   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
10055   \ignorespaces
10056 }

```

```

10057 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
10058 {
10059   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
10060   \ignorespaces
10061 }

10062 \keys_define:nn { nicematrix / Brace }
10063 {
10064   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
10065   left-shorten .value_forbidden:n = true ,
10066   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
10067   right-shorten .value_forbidden:n = true ,
10068   shorten .meta:n = { left-shorten , right-shorten } ,
10069   shorten .value_forbidden:n = true ,
10070   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
10071   color .tl_set:N = \l_tmpa_tl ,
10072   color .value_required:n = true ,
10073   unknown .code:n =
10074     \@@_unknown_key:nn
10075     { nicematrix / Brace }
10076     { Unknown-key-for-Brace }
10077 }

```

#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*.

```

10078 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
10079 {
10080   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

10081   \@@_compute_i_j:nn { #1 } { #2 }
10082   \bool_lazy_or:nnTF
10083     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
10084     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
10085     {
10086       \str_if_eq:eeTF { #5 } { under }
10087       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
10088       { \@@_error:nn { Construct-too-large } { \OverBrace } }
10089     }
10090     {
10091       \tl_clear:N \l_tmpa_tl
10092       \keys_set:nn { nicematrix / Brace } { #4 }
10093       \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
10094       \pgfpicture
10095       \pgfrememberpicturepositiononpagetrue
10096       \pgf@relevantforpicturesizefalse
10097       \bool_if:NT \l_@@_brace_left_shorten_bool
10098       {
10099         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
10100         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10101         {
10102           \cs_if_exist:cT
10103             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
10104             {
10105               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
10106
10107               \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
10108               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
10109             }
10110         }
10111       }
10112       \bool_lazy_or:nnT

```

```

10113 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
10114 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
10115 {
10116   \@@_qpoint:n { col - \l_@@_first_j_tl }
10117   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
10118 }
10119 \bool_if:NT \l_@@_brace_right_shorten_bool
10120 {
10121   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
10122   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10123   {
10124     \cs_if_exist:cT
10125     { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
10126     {
10127       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
10128       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
10129       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
10130     }
10131   }
10132 }
10133 \bool_lazy_or:nnT
10134 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
10135 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
10136 {
10137   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
10138   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
10139 }
10140 \pgfset { inner~sep = \c_zero_dim }
10141 \str_if_eq:eeTF { #5 } { under }
10142 { \@@_underbrace_i:n { #3 } }
10143 { \@@_overbrace_i:n { #3 } }
10144 \endpgfpicture
10145 }
10146 \group_end:
10147 }

```

The argument is the text to put above the brace.

```

10148 \cs_new_protected:Npn \@@_overbrace_i:n #1
10149 {
10150   \@@_qpoint:n { row - \l_@@_first_i_tl }
10151   \pgftransformshift
10152   {
10153     \pgfpoint
10154     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
10155     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
10156   }
10157   \pgfnode
10158   { rectangle }
10159   { south }
10160   {
10161     \vtop
10162     {
10163       \group_begin:
10164       \everycr { }
10165       \halign
10166       {
10167         \hfil ## \hfil \crcr
10168         \bool_if:NTF \l_@@_tabular_bool
10169         { \begin { tabular } { c } #1 \end { tabular } }
10170         { $ \begin { array } { c } #1 \end { array } $ }
10171         \cr
10172         $ % $
10173         \overbrace
10174         {

```

```

10175             \hbox_to_wd:nn
10176             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10177             { }
10178         }
10179         $ % $
10180         \cr
10181     }
10182     \group_end:
10183 }
10184 }
10185 { }
10186 { }
10187 }

```

The argument is the text to put under the brace.

```

10188 \cs_new_protected:Npn \@@_underbrace_i:n #1
10189 {
10190     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
10191     \pgftransformshift
10192     {
10193         \pgfpoint
10194         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
10195         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
10196     }
10197     \pgfnode
10198     { rectangle }
10199     { north }
10200     {
10201         \group_begin:
10202         \everycr { }
10203         \vbox
10204         {
10205             \halign
10206             {
10207                 \hfil ## \hfil \crcr
10208                 $ % $
10209                 \underbrace
10210                 {
10211                     \hbox_to_wd:nn
10212                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10213                     { }
10214                 }
10215                 $ % $
10216                 \cr
10217                 \bool_if:NTF \l_@@_tabular_bool
10218                 { \begin { tabular } { c } #1 \end { tabular } }
10219                 { $ \begin { array } { c } #1 \end { array } $ }
10220                 \cr
10221             }
10222         }
10223     \group_end:
10224 }
10225 { }
10226 { }
10227 }

```

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`).

```

10228 \AddToHook { package / tikz / after }
10229 {
10230   \tikzset
10231   {
10232     nicematrix / brace / .style =
10233     {
10234       decoration = { brace , raise = -0.15 em } ,
10235       decorate ,
10236     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

10237     nicematrix / mirrored-brace / .style =
10238     {
10239       nicematrix / brace ,
10240       decoration = mirror ,
10241     }
10242   }
10243 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

10244 \keys_define:nn { nicematrix / Hbrace }
10245 {
10246   color .code:n = ,
10247   horizontal-label .code:n = ,
10248   horizontal-labels .code:n = ,
10249   shorten .code:n = ,
10250   shorten-start .code:n = ,
10251   shorten-end .code:n = ,
10252   shorten+ .code:n = ,
10253   shorten-start+ .code:n = ,
10254   shorten-end+ .code:n = ,
10255   shorten~+ .code:n = ,
10256   shorten-start~+ .code:n = ,
10257   shorten-end~+ .code:n = ,
10258   brace-shift .code:n = ,
10259   brace-shift+ .code:n = ,
10260   brace-shift~+ .code:n = ,
10261   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
10262 }

```

Here we need an “fully expandable” command.

```

10263 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
10264 {
10265   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10266   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
10267   { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
10268 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

10269 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3

```

```

10270 {
10271   \int_compare:nNnTF \c@iRow < { 2 }
10272   {
We recall that \str_if_eq:nnTF is “fully expandable”.
10273     \str_if_eq:nnTF { #2 } { * }
10274     {
10275       \bool_set_true:N \l_@@_nullify_dots_bool
10276       \Ldots
10277       [
10278         line-style = nicematrix / brace ,
10279         #1 ,
10280         up =
10281         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10282       ]
10283     }
10284     {
10285       \Hdotsfor
10286       [
10287         line-style = nicematrix / brace ,
10288         #1 ,
10289         up =
10290         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10291       ]
10292       { #2 }
10293     }
10294   }
10295   {
10296     \str_if_eq:nnTF { #2 } { * }
10297     {
10298       \bool_set_true:N \l_@@_nullify_dots_bool
10299       \Ldots
10300       [
10301         line-style = nicematrix / mirrored-brace ,
10302         #1 ,
10303         down =
10304         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10305       ]
10306     }
10307     {
10308       \Hdotsfor
10309       [
10310         line-style = nicematrix / mirrored-brace ,
10311         #1 ,
10312         down =
10313         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10314       ]
10315       { #2 }
10316     }
10317   }
10318   \keys_set:nn { nicematrix / Hbrace } { #1 }
10319 }

```

```

10320 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
10321 {
10322   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10323   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10324   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
10325 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

10326 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3

```

```

10327 {
10328   \int_compare:nNnTF \c@jCol < { 2 }
10329   {
10330     \str_if_eq:nnTF { #2 } { * }
10331     {
10332       \bool_set_true:N \l_@@_nullify_dots_bool
10333       \Vdots
10334       [
10335         Vbrace ,
10336         line-style = nicematrix / mirrored-brace ,
10337         #1 ,
10338         down =
10339         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10340       ]
10341     }
10342     {
10343       \Vdotsfor
10344       [
10345         Vbrace ,
10346         line-style = nicematrix / mirrored-brace ,
10347         #1 ,
10348         down =
10349         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10350       ]
10351       { #2 }
10352     }
10353   }
10354   {
10355     \str_if_eq:nnTF { #2 } { * }
10356     {
10357       \bool_set_true:N \l_@@_nullify_dots_bool
10358       \Vdots
10359       [
10360         Vbrace ,
10361         line-style = nicematrix / brace ,
10362         #1 ,
10363         up =
10364         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10365       ]
10366     }
10367     {
10368       \Vdotsfor
10369       [
10370         Vbrace ,
10371         line-style = nicematrix / brace ,
10372         #1 ,
10373         up =
10374         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10375       ]
10376       { #2 }
10377     }
10378   }
10379   \keys_set:nn { nicematrix / Hbrace } { #1 }
10380 }

```

35 The command TikzEveryCell

```

10381 \bool_new:N \l_@@_not_empty_bool
10382 \bool_new:N \l_@@_empty_bool

```

```

10383
10384 \keys_define:nn { nicematrix / TikzEveryCell }
10385 {
10386   not-empty .code:n =
10387     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10388     { \bool_set_true:N \l_@@_not_empty_bool }
10389     { \@@_error:n { detection-of-empty-cells } } ,
10390   not-empty .value_forbidden:n = true ,
10391   empty .code:n =
10392     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10393     { \bool_set_true:N \l_@@_empty_bool }
10394     { \@@_error:n { detection-of-empty-cells } } ,
10395   empty .value_forbidden:n = true ,
10396   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
10397 }
10398
10399
10400 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10401 {
10402   \IfPackageLoadedTF { tikz }
10403   {
10404     \group_begin:
10405     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

10406     \tl_set:Nn \l_tmpa_tl { { #2 } }
10407     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10408     { \@@_for_a_block:nnnnn ##1 }
10409     \@@_all_the_cells:
10410     \group_end:
10411   }
10412   { \@@_error:n { TikzEveryCell~without~tikz } }
10413 }
10414
10415
10416 \cs_new_protected:Nn \@@_all_the_cells:
10417 {
10418   \int_step_inline:nn \c@iRow
10419   {
10420     \int_step_inline:nn \c@jCol
10421     {
10422       \cs_if_exist:cF { cell - ##1 - #####1 }
10423       {
10424         \@@_if_in_corner:nF { ##1 - #####1 }
10425         {
10426           \bool_set_false:N \l_tmpa_bool
10427           \cs_if_exist:cTF
10428           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10429           {
10430             \bool_if:NF \l_@@_empty_bool
10431             { \bool_set_true:N \l_tmpa_bool }
10432           }
10433           {
10434             \bool_if:NF \l_@@_not_empty_bool
10435             { \bool_set_true:N \l_tmpa_bool }
10436           }
10437           \bool_if:NT \l_tmpa_bool
10438           {
10439             \@@_block_tikz:onnnn
10440             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10441           }
10442         }
10443       }

```

```

10444     }
10445   }
10446 }
10447
10448 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10449 {
10450   \bool_if:NF \l_@@_empty_bool
10451   {
10452     \@@_block_tikz:nnnnn
10453     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10454   }
10455   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10456 }
10457
10458 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10459 {
10460   \int_step_inline:nnn { #1 } { #3 }
10461   {
10462     \int_step_inline:nnn { #2 } { #4 }
10463     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10464   }
10465 }

```

36 The key draw-trees-in-col

```

10466 \cs_new_protected:Npn \@@_draw_trees:
10467 {
10468   \bool_if:NTF \l_@@_no_cell_nodes_bool
10469   { \@@_error:n { draw-trees~with~no~cell~nodes } }
10470   \@@_draw_trees_i:
10471 }
10472
10473 \cs_new_protected:Npn \@@_draw_trees_i:
10474 {
10475   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10476   \dim_zero_new:N \l_@@_em_dim
10477   \dim_set:Nn \l_@@_em_dim { 1 em }
10478   \dim_zero_new:N \l_@@_ex_dim
10479   \dim_set:Nn \l_@@_ex_dim { 1 ex }
10480   \pgfpicture
10481   \pgfrememberpicturepositiononpagetrue
10482   \pgf@relevantforpicturesizefalse
10483   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10484   { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10485   \@@_color:o \l_@@_trees_color_tl
10486   \pgfsetcornersarced
10487   { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10488   \clist_map_function:NN \g_@@_col_with_trees_clist
10489   \@@_draw_trees_in_col:n
10490   \clist_gclear:N \g_@@_col_with_trees_clist
10491   \endpgfpicture
10492 }
10493
10494 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10495 {

```

The argument is provided by curryfication.

```

10494   \int_compare:nNnTF { #1 } > \c@jCol
10495   { \@@_error:nn { Col~outside~tabular~in~trees } }
10496   {
10497     \int_compare:nNnTF { #1 } = \c@jCol
10498     { \@@_error:nn { Last~col~in~trees } }

```

```

10499         \@@_draw_trees_in_col_i:n
10500     }
10501     { #1 }
10502 }

10503 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10504 {
10505     \int_set:Nn \l_tmpa_int { 1 }
10506     \int_step_inline:nn { \c@iRow + 1 }
10507     {
10508         \cs_if_exist:cT
10509         { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10510         {
10511             \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10512             {

```

Now, you will, potentially, draw a tree.

```

10513         \@@_draw_tree:nee
10514         { #1 }
10515         { \int_use:N \l_tmpa_int }
10516         { \int_eval:n { ##1 - 1 } }
10517     }
10518     \int_set:Nn \l_tmpa_int { ##1 }
10519 }
10520 }
10521 \int_compare:nNnT \c@iRow > \l_tmpa_int
10522 {
10523     \@@_draw_tree:nee
10524     { #1 }
10525     { \int_use:N \l_tmpa_int }
10526     { \int_eval:n { \c@iRow } }
10527 }
10528 }

```

#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.

```

10529 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10530 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10531     \pgfpointanchor { \@@_env: - #1 } { 5 }
10532     \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.

```

10533     \bool_set_false:N \l_tmpa_bool
10534     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10535     {
10536         \cs_if_exist:cT
10537         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10538     {
10539         \pgfpointanchor
10540         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10541         { base~west }
10542         \pgfpathmoveto
10543         {
10544             \pgfpoint
10545             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10546             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10547         }
10548         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10549         \bool_if:NF \l_tmpa_bool
10550         {

```

```

10551         \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10552         \pgfpathlineto
10553         { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10554         \bool_set_true:N \l_tmpa_bool
10555     }
10556     \pgfusepath { stroke }
10557 }
10558 }
10559 }
10560 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10561 \cs_new_protected:Npn \@@_create_blocks_in_col:
10562 {
10563     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10564     \clist_map_inline:Nn \g_@@_cbic_clist
10565     {
10566         \cs_set:cpn
10567         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { \c@iRow + 1 } - ##1 }
10568         { rien }

```

`\l_tmpa_int` will be the first row of the block being created.

```

10569     \int_set:Nn \l_tmpa_int { 1 }
10570     \int_step_inline:nn { \c@iRow + 1 }
10571     {
10572         \bool_set_false:N \l_tmpa_bool
10573         \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10574         { \bool_set_true:N \l_tmpa_bool }
10575         \clist_if_in:NnT \g_@@_false_rows_clist { #####1 }
10576         { \bool_set_true:N \l_tmpa_bool }
10577         \bool_if:NT \l_tmpa_bool
10578         {
10579             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10580             {
10581                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10582                 {
10583                     { \int_use:N \l_tmpa_int }
10584                     { ##1 }
10585                     { \int_eval:n { #####1 - 1 } }
10586                     { ##1 }
10587                     { }
10588                 }
10589             }
10590             \int_set:Nn \l_tmpa_int { #####1 }
10591         }
10592     }
10593 }
10594 \clist_gclear:N \g_@@_cbic_clist
10595 }

```

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.

```

10596 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10597 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

```

```

10598 \NewDocumentCommand \@@_ShowCellNames { }
10599 {
10600   \bool_if:NT \l_@@_in_code_after_bool
10601   {
10602     \pgfpicture
10603     \pgfrememberpicturepositiononpagetrue
10604     \pgf@relevantforpicturesizefalse
10605     \pgfpathrectanglecorners
10606       { \@@_qpoint:n { 1 } }
10607       { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10608     \pgfsetfillopacity { 0.75 }
10609     \pgfsetfillcolor { white }
10610     \pgfusepathqfill
10611     \endpgfpicture
10612   }
10613   \dim_gzero_new:N \g_@@_tmpc_dim
10614   \dim_gzero_new:N \g_@@_tmpd_dim
10615   \dim_gzero_new:N \g_@@_tmppe_dim
10616   \int_step_inline:nn \c@iRow
10617   {
10618     \clist_if_in:NnF \g_@@_false_rows_clist { ##1 }
10619     { \@@_show_cell_names_one_row:n { ##1 } }
10620   }
10621 }

10622 \cs_new_protected:Npn \@@_show_cell_names_one_row:n #1
10623 {
10624   \bool_if:NTF \l_@@_in_code_after_bool
10625   {
10626     \pgfpicture
10627     \pgfrememberpicturepositiononpagetrue
10628     \pgf@relevantforpicturesizefalse
10629   }
10630   { \begin { pgfpicture } }
10631   \@@_qpoint:n { row - #1 }
10632   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10633   \@@_qpoint:n { row - \int_eval:n { #1 + 1 } }
10634   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10635   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10636   \bool_if:NTF \l_@@_in_code_after_bool
10637   { \endpgfpicture }
10638   { \end { pgfpicture } }
10639   \int_step_tokens:nn \c@jCol
10640   { \@@_show_cell_names_one_cell:nn { #1 } }
10641 }

10642 \cs_new_protected:Npn \@@_show_cell_names_one_cell:nn #1 #2
10643 {
10644   \clist_if_in:NnF \g_@@_false_cols_clist { #2 }
10645   { \@@_show_cell_names_one_cell_i:nn { #1 } { #2 } }
10646 }

10647 \cs_new_protected:Npn \@@_show_cell_names_one_cell_i:nn #1 #2
10648 {
10649   \hbox_set:Nn \l_tmpa_box
10650   {
10651     \normalfont \Large \sffamily \bfseries
10652     \bool_if:NTF \l_@@_in_code_after_bool
10653     { \color { red } }
10654     { \color { red ! 50 } }
10655     #1 - #2
10656   }
10657   \bool_if:NTF \l_@@_in_code_after_bool
10658   {

```

```

10659     \pgfpicture
10660     \pgfrememberpicturepositiononpagetrue
10661     \pgf@relevantforpicturesizefalse
10662   }
10663   { \begin { pgfpicture } }
10664   \@_qpoint:n { col - #2 }
10665   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10666   \@_qpoint:n { col - \int_eval:n { #2 + 1 } }
10667   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10668   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10669   \bool_if:NTF \l_@@_in_code_after_bool
10670     { \endpgfpicture }
10671     { \end { pgfpicture } }
10672   \fp_set:Nn \l_tmpa_fp
10673   {
10674     \fp_min:nn
10675     {
10676       \fp_min:nn
10677         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10678         { \dim_ratio:nn \g_@@_tmpe_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10679     }
10680     { 1.0 }
10681   }
10682   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10683   \pgfpicture
10684   \pgfrememberpicturepositiononpagetrue
10685   \pgf@relevantforpicturesizefalse
10686   \pgftransformshift
10687   {
10688     \pgfpoint
10689       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10690     \g_@@_tmpa_dim
10691   }
10692   \pgfnode
10693     { rectangle }
10694     { center }
10695     { \box_use:N \l_tmpa_box }
10696     { }
10697     { }
10698   \endpgfpicture
10699 }

```

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.

```
10700 \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.

```
10701 \bool_new:N \g_@@_footnote_bool
```

```

10702 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10703 {
10704   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10705   but-that-key-is-unknown. \\
10706   It-will-be-ignored. \\
10707   For-a-list-of-the-available-keys,~type-H-<return>.
10708 }
10709 {
10710   The-available-keys-are~(in~alphabetic~order):~
10711   footnote,~
10712   footnotehyper,~
10713   messages-for-Overleaf,~
10714   renew-dots-and~
10715   renew-matrix.
10716 }
10717 \keys_define:nn { nicematrix }
10718 {
10719   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10720   renew-dots .value_forbidden:n = true ,
10721   renew-matrix .code:n = \@@_renew_matrix: ,
10722   renew-matrix .value_forbidden:n = true ,
10723   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10724   footnote .bool_set:N = \g_@@_footnote_bool ,
10725   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10726   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10727 }
10728 \ProcessKeyOptions

10729 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10730 {
10731   You-can't-use-the-option~'footnote'~because-the-package~
10732   footnotehyper~has-already-been-loaded.~
10733   If-you-want,~you-can-use-the-option~'footnotehyper'~and-the-footnotes~
10734   within~the~environments~of~nicematrix~will-be~extracted~with~the~tools~
10735   of~the~package~footnotehyper.\\
10736   The~package~footnote~won't-be-loaded.
10737 }
10738 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10739 {
10740   You-can't-use-the-option~'footnotehyper'~because-the-package~
10741   footnote~has-already-been-loaded.~
10742   If-you-want,~you-can-use-the-option~'footnote'~and-the-footnotes~
10743   within~the~environments~of~nicematrix~will-be~extracted~with~the~tools~
10744   of~the~package~footnote.\\
10745   The~package~footnotehyper~won't-be-loaded.
10746 }

10747 \bool_if:NT \g_@@_footnote_bool
10748 {
10749   \IfClassLoadedTF { beamer }
10750   { \bool_set_false:N \g_@@_footnote_bool }
10751   {
10752     \IfPackageLoadedTF { footnotehyper }
10753     { \@@_error:n { footnote~with~footnotehyper~package } }
10754     { \usepackage { footnote } }
10755   }
10756 }

10757 \bool_if:NT \g_@@_footnotehyper_bool
10758 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

10759 \IfClassLoadedTF { beamer }
10760 { \bool_set_false:N \g_@@_footnote_bool }
10761 {
10762   \IfPackageLoadedTF { footnote }
10763   { \@@_error:n { footnotehyper~with~footnote~package } }
10764   { \usepackage { footnotehyper } }
10765 }
10766 \bool_set_true:N \g_@@_footnote_bool
10767 }

```

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.

```

10768 \IfPackageLoadedF { underscore }
10769 {
10770   \AddToHook { package / underscore / after }
10771   { \@@_error:n { underscore~after~nicematrix } }
10772 }

```

41 Compatibility with threeparttable

```

10773 \AddToHook { package / threeparttable / after }
10774 {
10775   \AddToHook { env / threeparttable / begin }
10776   {
10777     \TPT@hookin { NiceTabular }
10778     \TPT@hookin { NiceTabular* }
10779     \TPT@hookin { NiceTabularX }
10780   }
10781 }

```

42 Gestion of the errors

42.1 Security in the CodeBefore and the CodeAfter

```

10782 \cs_new_protected:Npn \@@_security_font_commands:
10783 {
10784   \cs_set_eq:NN \@@_old_small: \small
10785   \cs_set_eq:NN \@@_old_footnotesize: \footnotesize
10786   \cs_set_eq:NN \@@_old_scriptsize: \scriptsize
10787   \cs_set_eq:NN \@@_old_tiny: \tiny
10788   \cs_set_eq:NN \small \@@_small:
10789   \cs_set_eq:NN \footnotesize \@@_footnotesize:
10790   \cs_set_eq:NN \scriptsize \@@_scriptsize:
10791   \cs_set_eq:NN \tiny \@@_tiny:
10792   \everyhbox
10793   {

```

```

10794     \cs_set_eq:NN \small \@@_old_small:
10795     \cs_set_eq:NN \footnotesize \@@_old_footnotesize:
10796     \cs_set_eq:NN \scriptsize \@@_old_scriptsize:
10797     \cs_set_eq:NN \tiny \@@_old_tiny:
10798   }
10799 }

10800 \cs_set_protected:Npn \@@_small: { \@@_font_command_error:N \small }
10801 \cs_set_protected:Npn \@@_footnotesize: { \@@_font_command_error:N \footnotesize }
10802 \cs_set_protected:Npn \@@_scriptsize: { \@@_font_command_error:N \scriptsize }
10803 \cs_set_protected:Npn \@@_tiny: { \@@_font_command_error:N \tiny }

10804 \cs_new_protected:Npn \@@_font_command_error:N #1
10805 { \@@_error:nn { font-command } { #1 } }

10806 \@@_msg_new:nn { font-command }
10807 {
10808   Font-command-not-allowed.\@
10809   You~can't~use~the~command~#1 directly~in~the~
10810   \bool_if:NTF \l_@@_in_code_after_bool
10811     { \token_to_str:N \CodeAfter}
10812     { \token_to_str:N \CodeBefore}.~Your~command~will~be~ignored.
10813 }

```

42.2 The 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.

```

10814 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10815 {
10816   \str_if_eq:eeTF
10817     { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10818     { + }
10819     {
10820       \str_set:Ne \l_tmpa_str
10821       { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10822       \bool_set_false:N \l_tmpa_bool
10823       \clist_map_inline:nn { #1 }
10824       {
10825         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10826         {
10827           \@@_error:n { key~without~+~exists }
10828           \bool_set_true:N \l_tmpa_bool
10829           \clist_map_break:
10830         }
10831       }
10832       \bool_if:NF \l_tmpa_bool
10833       {
10834         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10835         \@@_unknown_key_i:nn { #1 } { #2 }
10836       }
10837     }
10838     { \@@_unknown_key_i:nn { #1 } { #2 } }
10839 }

```

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.

```

10840 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10841 {

```

```

10842 \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10843 \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10844 \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10845 \bool_set_false:N \l_tmpa_bool
10846 \clist_map_inline:nn { #1 }
10847 {
10848   \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10849   {
10850     \@@_error:n { key~with~normal~form~exists }
10851     \bool_set_true:N \l_tmpa_bool
10852     \clist_map_break:
10853   }
10854 }
10855 \bool_if:NF \l_tmpa_bool
10856 {
10857   \@@_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).

```

10858   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10859   { \msg_info:nn { nicematrix } { #2~+ } }
10860 }
10861 }
10862 \@@_msg_new:nn { key~without~+~exists }
10863 {
10864   Problem~with~a~key.\\
10865   The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10866   additive~syntax~(with~+=).~It~will~be~ignored.
10867 }
10868 \@@_msg_new:nn { key~with~normal~form~exists }
10869 {
10870   No~key~'\l_keys_key_str'.\\
10871   It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10872 }
10873 \str_const:Ne \c_@@_available_keys_str
10874 {
10875   \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10876   { For~a~list~of~the~available~keys,~type~H~<return>. }
10877 }
10878 \seq_new:N \g_@@_types_of_matrix_seq
10879 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10880 {
10881   NiceMatrix ,
10882   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10883 }
10884 \seq_gset_map_e:Nnn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10885 { \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`.

```

10886 \cs_new_protected:Npn \@@_err_too_many_cols:
10887 {
10888   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10889   { \@@_fatal:nn { too-many-cols-for-array } }
10890   \int_compare:nNnT \l_@@_last_col_int = { -2 }
10891   { \@@_fatal:n { too-many-cols-for-matrix } }
10892   \int_compare:nNnT \l_@@_last_col_int = { -1 }

```

```

10893     { \@@_fatal:n { too-many-cols-for-matrix } }
10894     \bool_if:NF \l_@@_last_col_without_value_bool
10895     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10896 }

```

The following command must *not* be protected since it's used in an error message.

```

10897 \cs_new:Npn \@@_message_hdotsfor:
10898 {
10899     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10900     { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
10901       \token_to_str:N \Hbrace \ is-incorrect. }
10902 }

10903 \cs_new_protected:Npn \@@_Hline_in_cell:
10904 { \@@_error:n { Misuse-of-Hline } }

10905 \@@_msg_new:nn { Misuse-of-Hline }
10906 {
10907     Misuse-of~\token_to_str:N \Hline. \\
10908     Error-in-the-row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:~
10909     \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must-be-used-only~
10910     at-the-beginning-of-a-row.~Maybe-you-have-forgotten-a~\token_to_str:N \\~
10911     Your-command-will-be-ignored.
10912 }

10913 \cs_new_protected:Npn \@@_hdashedline:
10914 { \@@_error:n { Misuse-of-hdashedline } }
10915 \cs_new_protected:Npn \@@_cdashedline:n #1
10916 { \@@_error:n { Misuse-of-cdashedline } }

10917 \@@_msg_new:nn { Misuse-of-hdashedline }
10918 {
10919     You-should-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\})~in-order~
10920     to-be-able-to-use-the-command~\token_to_str:N \hdashedline.~Your-command~
10921     will-be-ignored.
10922 }
10923 \@@_msg_new:nn { Misuse-of-cdashedline }
10924 {
10925     You-should-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\})~in-order~
10926     to-be-able-to-use-the-command~\token_to_str:N \cdashedline.~Your-command~
10927     will-be-ignored.
10928 }

10929 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10930 {
10931     Incompatible-options.\\
10932     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-the-same-time.~
10933     The-output-will-not-be-reliable.
10934 }

10935 \@@_msg_new:nn { Body-alone }
10936 {
10937     \token_to_str:N \Body\ alone. \\
10938     You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10939 }

10940 \@@_msg_new:nn { Misuse-of-CodeBefore }
10941 {
10942     Misuse-of~\token_to_str:N \CodeBefore. \\
10943     \token_to_str:N \CodeBefore\ must-be-used-only-at-the-beginning-of~
10944     the-environment.
10945 }

10946 \@@_msg_new:nn { NiceTabularX-probably-required }
10947 {
10948     Incorrect-syntax.\\
10949     You-probably-want-to-use~\{NiceTabularX\}~whose-first-argument-is-the~

```

```

10950 ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10951 with~the~key~'width'.
10952 }

10953 \@@_msg_new:nn { cellcolor~in~Block }
10954 {
10955   Misuse~of~\token_to_str:N \cellcolor \\
10956   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10957   \bool_if:NTF \l_@@_amp_in_blocks_bool
10958     { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10959     { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10960     .~Here,~you~should~use~the~key~'fill'~of~the~block.~Your~command~will~be~ignored.
10961 }

10962 \@@_msg_new:nn { rowcolor~in~Block }
10963 {
10964   Misuse~of~\token_to_str:N \rowcolor \\
10965   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10966   However,~it's~possible~to~color~the~block~with~its~key~'fill'.~
10967   Your~command~will~be~ignored.
10968 }

10969 \@@_msg_new:nn { key~color~inside }
10970 {
10971   Deleted~key.\\
10972   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10973   ~'nicematrix'~and~must~not~be~used.
10974 }

10975 \@@_msg_new:nn { Misuse~of~FalseRow }
10976 {
10977   Misuse~of~\token_to_str:N \FalseRow \\
10978   Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:.~
10979   \token_to_str:N \FalseRow\ must~be~used~only~at~the~beginning~of~a~row.~
10980   Your~command~will~be~ignored.
10981 }

10982 \@@_msg_new:nn { Invalid~argument~for~w }
10983 {
10984   Invalid~argument~for~w.\\
10985   You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10986   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10987   If~you~go~on,~'c'~will~be~used.
10988 }

10989 \@@_msg_new:nn { invalid~weight }
10990 {
10991   Unknown~key.\\
10992   The~key~'\l_keys_key_str'~of~your~column~X~is~unknown~and~will~be~ignored.~
10993   The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10994   \IfPackageLoadedTF { varwidth }
10995     { (since~'varwidth'~is~loaded)~}
10996     { (if~you~load~'varwidth')~}
10997   and~real~numbers~for~the~weight~of~the~X~column.
10998 }

10999 \@@_msg_new:nn { last~col~not~used }
11000 {
11001   Column~not~used.\\
11002   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
11003   in~your~\@@_full_name_env:.~However,~you~can~go~on.
11004 }

11005 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
11006 {
11007   Too~many~columns.\\
11008   In~the~row~ \int_eval:n { \c@iRow },~
11009   you~try~to~use~more~columns~

```

```

11010     than-allowed-by-your~ \@@_full_name_env: .
11011     \@@_message_hdotsfor: \
11012     The-maximal-number-of-columns-is~ \int_eval:n { \l_@@_last_col_int - 1 }~
11013     (plus-the-exterior-columns).~
11014     Maybe-you-have-forgotten-a~\token_to_str:N \\.
11015   }
11016   \@@_msg_new:nn { too-many-cols-for-matrix }
11017   {
11018     Too-many-columns.\\
11019     In-the-row~ \int_eval:n { \c@iRow } ,~
11020     you-try-to-use-more-columns-than-allowed-by-your~ \@@_full_name_env: .
11021     \@@_message_hdotsfor: \
11022     Recall-that-the-maximal-number-of-columns-for-a-matrix~
11023     (excepted-the-potential-exterior-columns)-is-fixed-by-the~
11024     LaTeX-counter~'MaxMatrixCols'.~
11025     Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
11026     (use~ \token_to_str:N \setcounter \ to-change-that-value).~
11027     Maybe,~you-have-forgotten-a~\token_to_str:N \\.
11028   }

11029   \cs_set:Npn \@@_message_about_false_cols:
11030   {
11031     \int_compare:nNnTF { \clist_count:N \g_@@_false_cols_clist } = 1
11032     { (including-one-false-column-F)~ }
11033     {
11034       \int_compare:nNnT { \clist_count:N \g_@@_false_cols_clist } > 1
11035       {
11036         (including~\int_to_arabic:n { \clist_count:N \g_@@_false_cols_clist }~
11037         false-columns-F)~
11038       }
11039     }
11040   }

11041   \@@_msg_new:nn { too-many-cols-for-array }
11042   {
11043     Too-many-columns.\\
11044     In-the-row~ \int_eval:n { \c@iRow } ,~
11045     ~you-try-to-use-more-columns-than-allowed-by-your~
11046     \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
11047     \int_use:N \g_@@_static_num_of_col_int \
11048     \@@_message_about_false_cols:
11049     \bool_if:nT
11050     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
11051     { (plus-the-exterior-ones)-}
11052     since-the-preamble-is~' \g_@@_user_preamble_tl '~
11053     Maybe,~you-have-forgotten-a~\token_to_str:N \\.
11054   }

11055   \@@_msg_new:nn { columns-not-used }
11056   {
11057     Columns-not-used.\\
11058     The-preamble-of-your~ \@@_full_name_env: \ is~
11059     '\exp_not:V \g_@@_user_preamble_tl'.~
11060     It-announces~ \int_use:N \g_@@_static_num_of_col_int \
11061     columns~\@@_message_about_false_cols:
11062     but-you-only-used~ \int_use:N \c@jCol .~The-columns-that-you-did-not~
11063     use-won't-be-created.~You-won't-have-similar-warning-till-the-end-of-the-document.
11064   }

11065   \str_const:Nn \c_@@_explanation_no_cell_nodes_str
11066   {
11067     because-the-key~'no-cell-nodes'~is-in-force~
11068     (you-should-deactive-the-key~'no-cell-nodes'~whose-only-goal~
11069     is-to-speed-compilation-up)
11070   }

```

```

11071 \@@_msg_new:nn { xdots-without-cell-nodes }
11072 {
11073   Command~#1 forbidden.\\
11074   You~can't~use~the~command~#1~\c_@@_explanation_no_cell_nodes_str.~
11075   Your~command~will~be~ignored.
11076 }
11077 \@@_msg_new:nn { Misuse-of-NiceTabularNotes }
11078 {
11079   Misuse~of~\token_to_str:N \NiceTabularNotes \\
11080   \token_to_str:N\NiceTabularNotes\ should~be~used~only~after~at~tabular~
11081   which~uses~\notes/no-print`.~ Your~command~will~be~ignored.
11082 }
11083 \@@_msg_new:nn { empty-preamble }
11084 {
11085   Empty~preamble.\\
11086   The~preamble~of~your~ \@@_full_name_env: \ is~empty.
11087 }
11088 \@@_msg_new:nn { in-first-col }
11089 {
11090   Misuse~of~#1\\
11091   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.~
11092   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-col'.
11093 }
11094 \@@_msg_new:nn { in-last-col }
11095 {
11096   Misuse~of~#1\\
11097   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.~
11098   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-col'.
11099 }
11100 \@@_msg_new:nn { in-first-row }
11101 {
11102   Misuse~of~#1\\
11103   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
11104   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first-row'.
11105 }
11106 \@@_msg_new:nn { in-last-row }
11107 {
11108   Misuse~of~#1\\
11109   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
11110   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last-row'.
11111 }
11112 \@@_msg_new:nn { Row-outside-the-array }
11113 {
11114   Row~outside~the~array \\
11115   You~try,~with~the~key~'hlines',~to~draw~an~horizontal~rule~
11116   at~the~position~\int_use:N \l_@@_position_int\ but~it's~outside~the~array.~
11117   It~will~be~ignored.
11118 }
11119 \@@_msg_new:nn { Column-outside-the-array }
11120 {
11121   Column~outside~the~array \\
11122   You~try,~with~the~key~'vlines',~to~draw~a~vertical~rule~at~the~position~
11123   \int_use:N \l_@@_position_int\ but~it's~outside~the~array.~
11124   It~will~be~ignored.
11125 }
11126 \@@_msg_new:nn { TopRule-without-booktabs }
11127 {
11128   Misuse~of~#1\\
11129   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
11130   You~should~load~'booktabs'~(before~or~after~'nicematrix').~

```

```

11131     Your~command~will~be~ignored.
11132 }
11133 \@@_msg_new:nn{ rotate~in~p~col }
11134 {
11135     \token_to_str:N \rotate\ forbidden.\
11136     You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
11137     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~
11138     If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
11139     consider~the~classical~command~\token_to_str:N \rotatebox.
11140 }
11141 \@@_msg_new:nn { Not~empty~cell~in~false~column }
11142 {
11143     Not~empty~cell.\
11144     The~cell~(#{1-#2})~of~your~\@@_full_name_env: \
11145     is~not~empty~although~the~column~#2~is~a~
11146     "false~column"~(letter~F~in~the~preamble).~
11147     \bool_if:NTF \l_@@_light_syntax_bool
11148     { You~should~put~\{\}~for~an~empty~cell. }
11149     {
11150         \int_compare:nNnT { #2 } > 1
11151         { Recall~that~you~have~to~put~*two*~ampersands~(&&). }
11152     }
11153 }
11154 \@@_msg_new:nn { TopRule~without~tikz }
11155 {
11156     Misuse~of~#1\
11157     You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.~
11158     You~should~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\}).~
11159     \IfPackageLoadedF { booktabs }
11160     { You~should~also~load~'booktabs'~
11161       with~\token_to_str:N \usepackage \{booktabs\}.~ }
11162     Your~command~will~be~ignored.
11163 }
11164 \@@_msg_new:nn { caption~outside~float }
11165 {
11166     Key~caption~forbidden.\
11167     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
11168     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
11169 }
11170 \@@_msg_new:nn { short~caption~without~caption }
11171 {
11172     You~should~not~use~the~key~'short~caption'~without~'caption'.~
11173     However,~your~'short~caption'~will~be~used~as~'caption'.
11174 }
11175 \@@_msg_new:nn { double~closing~delimiter }
11176 {
11177     Double~delimiter.\
11178     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
11179     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
11180     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
11181 }
11182 \@@_msg_new:nn { delimiter~after~opening }
11183 {
11184     Double~delimiter.\
11185     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
11186     delimiter.~That~delimiter~will~be~ignored.
11187 }
11188 \@@_msg_new:nn { bad~option~for~line~style }
11189 {
11190     Bad~line~style.\

```

```

11191     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line-style'~
11192     is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
11193     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
11194 }
11195 \@@_msg_new:nn { draw-trees~with~no~cell-nodes }
11196 {
11197     Incompatible~keys.\\
11198     You~can't~use~the~key~'draw-trees-in-col'~
11199     here~\c_@@_explanation_no_cell_nodes_str.~
11200     If~you~go~on,~that~key~will~be~ignored.
11201 }
11202 \@@_msg_new:nn { corners~with~no~cell-nodes }
11203 {
11204     Incompatible~keys.\\
11205     You~can't~use~the~key~'corners'~here~\c_@@_explanation_no_cell_nodes_str.~
11206     If~you~go~on,~that~key~will~be~ignored.
11207 }
11208 \@@_msg_new:nn { extra-nodes~with~no~cell-nodes }
11209 {
11210     Incompatible~keys.\\
11211     You~can't~create~'extra-nodes'~here~\c_@@_explanation_no_cell_nodes_str.~
11212     If~you~go~on,~those~extra~nodes~won't~be~created.
11213 }
11214 \@@_msg_new:nn { Identical~notes~in~caption }
11215 {
11216     Identical~tabular~notes.\\
11217     You~can't~put~several~notes~with~the~same~content~in~
11218     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
11219     If~you~go~on,~the~output~will~probably~be~erroneous.
11220 }
11221 \@@_msg_new:nn { tabularnote~below~the~tabular }
11222 {
11223     \token_to_str:N \tabularnote \ forbidden\\
11224     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
11225     of~your~tabular~because~the~caption~will~be~composed~below~
11226     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
11227     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
11228     Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
11229     no~similar~error~will~raised~in~this~document.
11230 }
11231 \@@_msg_new:nn { Unknown~key~for~rules }
11232 {
11233     Unknown~key.\\
11234     There~are~only~three~keys~available~here:~'width',~'color'~and~
11235     'fix-vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
11236 }
11237 \@@_msg_new:nn { Unknown~key~for~trees }
11238 {
11239     Unknown~key.\\
11240     There~are~only~three~keys~available~here:~width~color~and~
11241     rounded-corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
11242 }
11243 % \end{macrocode}
11244 %
11245 %
11246 % \begin{macrocode}
11247 \@@_msg_new:nn { Unknown~key~for~Hbrace }
11248 {
11249     Unknown~key.\\
11250     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
11251     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \

```

```

11252     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
11253     'horizontal-label(s)',~'shorten'~'shorten-end'~
11254     and~'shorten-start'.
11255 }
11256 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
11257 {
11258     Unknown~key.\\
11259     There~is~only~two~keys~available~here:~
11260     'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
11261 }
11262 \@@_msg_new:nn { Unknown~key~for~rotate }
11263 {
11264     Unknown~key.\\
11265     The~only~keys~available~here~are~'c'~and~'-90'.~
11266     Your~key~' \l_keys_key_str '~will~be~ignored.
11267 }
11268 \@@_msg_new:nnn { Unknown~key~for~custom~line }
11269 {
11270     Unknown~key.\\
11271     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
11272     It~you~go~on,~you~will~probably~have~other~errors. \\
11273     \c_@@_available_keys_str
11274 }
11275 {
11276     The~available~keys~are~(in~alphabetic~order):~
11277     ccommand,~
11278     color,~
11279     dashed \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11280     command,~
11281     dotted,~
11282     letter,~
11283     multiplicity,~
11284     sep-color,~
11285     tikz \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11286     and~total~width.
11287 }
11288 \@@_msg_new:nnn { Unknown~key~for~default~line }
11289 {
11290     Unknown~key.\\
11291     The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
11292     It~you~go~on,~you~will~probably~have~other~errors. \\
11293     \c_@@_available_keys_str
11294 }
11295 {
11296     The~available~keys~are~(in~alphabetic~order):~
11297     color,~
11298     dashed \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11299     dotted,~
11300     multiplicity,~
11301     sep-color,~
11302     tikz~\IfPackageLoadedF { tikz } { (when~TikZ~is~loaded)~}
11303     and~total~width.
11304 }
11305 \@@_msg_new:nn { dashed~for~default~line~without~TikZ}
11306 {
11307     Key~'dashed'~forbidden.\\
11308     If~you~want~to~use~the~value~'dashed'~for~'default~line',~
11309     you~must~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\}).
11310 }
11311 \@@_msg_new:nnn { Unknown~key~for~xdots }
11312 {

```

```

11313     Unknown~key.\\
11314     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
11315     \c_@@_available_keys_str
11316 }
11317 {
11318     The~available~keys~are~(in~alphabetic~order):~
11319     'color',~
11320     'horizontal(s)-labels',~
11321     'inter',~
11322     'line-style',~
11323     'nullify',~
11324     'radius',~
11325     'shorten',~
11326     'shorten-end'~and~'shorten-start'.
11327 }
11328 \@@_msg_new:nn { Unknown~key~for~rowcolors }
11329 {
11330     Unknown~key.\\
11331     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
11332     (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
11333 }
11334 \@@_msg_new:nn { Col~outside~tabular~in~trees }
11335 {
11336     Error~with~'draw-trees-in-col' \\
11337     The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
11338     is~\int_use:N \c@jCol.~It~will~be~ignored.
11339 }
11340 \@@_msg_new:nn { Last~col~in~trees }
11341 {
11342     Error~with~'draw-trees-in-col' \\
11343     You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
11344     the~last~column~of~your~tabular.~It~will~be~ignored.
11345 }
11346 \@@_msg_new:nn { label~without~caption }
11347 {
11348     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
11349     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
11350 }
11351 \@@_msg_new:nn { W~warning }
11352 {
11353     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
11354     (row~ \int_use:N \c@iRow ).
11355 }
11356 \@@_msg_new:nn { Construct~too~large }
11357 {
11358     Construct~too~large.\\
11359     Your~command~ \token_to_str:N #1
11360     can't~be~drawn~because~your~matrix~is~too~small.~
11361     Your~command~will~be~ignored.
11362 }
11363 \@@_msg_new:nn { underscore~after~nicematrix }
11364 {
11365     Problem~with~'underscore'.\\
11366     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
11367     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
11368     ' \token_to_str:N \Cdots \token_to_str:N _
11369     \{ n \token_to_str:N \text \{ ~times \} \}'.
11370 }
11371 \@@_msg_new:nn { ampersand~in~light~syntax }
11372 {

```

```

11373     Ampersand~forbidden.\\
11374     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
11375     ~the~key~'light-syntax'~is~in~force.
11376 }
11377 \@@_msg_new:nn { double-backslash~in~light-syntax }
11378 {
11379     Double~backslash~forbidden.\\
11380     You~can't~use~ \token_to_str:N \\
11381     ~to~separate~rows~because~the~key~'light-syntax'~
11382     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
11383     (set~by~the~key~'end-of-row').
11384 }
11385 \@@_msg_new:nn { bad~value~for~baseline }
11386 {
11387     Bad~value~for~baseline.\\
11388     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11389     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
11390     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
11391     the~form~'line-i'.~A~value~of~1~will~be~used.
11392 }
11393 \@@_msg_new:nn { bad~value~for~baseline~line }
11394 {
11395     Bad~value~for~baseline~with~line.\\
11396     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11397     valid.~The~number~of~the~line~must~be~between~1~and~
11398     \int_eval:n { \c_iRow + 1 }.~
11399     A~value~of~'line-1'~will~be~used.
11400 }
11401 \@@_msg_new:nn { detection~of~empty~cells }
11402 {
11403     Problem~with~'not~empty'\\
11404     For~technical~reasons,~you~must~activate~
11405     'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
11406     in~order~to~use~the~key~' \l_keys_key_str '~
11407     Your~key~will~be~ignored.
11408 }
11409 \@@_msg_new:nn { siunitx~too~old }
11410 {
11411     siunitx~too~old\\
11412     You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
11413     is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26)~and~
11414     the~version~that~you~have~loaded~has~the~date:~
11415     \str_range:cn { ver @ siunitx . sty } { 1 } { 10 }.
11416 }
11417 \@@_msg_new:nn { Invalid~name }
11418 {
11419     Invalid~name.\\
11420     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
11421     \SubMatrix \ of~your~ \@@_full_name_env: .~
11422     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
11423     Your~key~will~be~ignored.
11424 }
11425 \@@_msg_new:nn { Hbrace~not~allowed }
11426 {
11427     Command~not~allowed.\\
11428     You~can't~use~the~command~ \token_to_str:N #1
11429     because~you~have~not~loaded~
11430     \IfPackageLoadedTF { tikz }
11431     { the~TikZ~library~'decorations.pathreplacing'.~Use~ }
11432     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
11433     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\

```

```

11434     Your~command~will~be~ignored.
11435 }

11436 \@@_msg_new:nn { Vbrace~not~allowed }
11437 {
11438     Command~not~allowed.\\
11439     You~can't~use~the~command~ \token_to_str:N \Vbrace \
11440     because~you~have~not~loaded~TikZ~
11441     and~the~TikZ~library~'decorations.pathreplacing'.~
11442     Use: ~\token_to_str:N \usepackage \{tikz\}~
11443     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
11444     Your~command~will~be~ignored.
11445 }

11446 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
11447 {
11448     Wrong~line.\\
11449     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
11450     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11451     number~is~not~valid.~It~will~be~ignored.
11452 }

11453 \@@_msg_new:nn { Impossible~SubMatrix }
11454 {
11455     Impossible~SubMatrix.\\
11456     It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
11457     because~all~the~cells~are~empty~in~the~column~on~the~#1~
11458     side~of~that~\token_to_str:N \SubMatrix.
11459     \bool_if:NT \l_@@_submatrix_slim_bool
11460     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
11461     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11462 }

11463 \@@_msg_new:nn { Impossible~SubMatrix~no~cell~nodes }
11464 {
11465     Impossible~SubMatrix.\\
11466     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
11467     \c_@@_explanation_no_cell_nodes_str.~
11468     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11469 }

11470 \@@_msg_new:nnn { width~without~X~columns }
11471 {
11472     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11473     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11474     Your~key~will~be~ignored.
11475 }
11476 {
11477     This~message~is~the~message~'width~without~X~columns'~
11478     of~the~module~'nicematrix'.~
11479     The~experimented~users~can~disable~that~message~with~
11480     \token_to_str:N \msg_redirect_name:nnn .\\
11481 }

11482

11483 \@@_msg_new:nn { key~multiplicity~with~dotted }
11484 {
11485     Incompatible~keys. \\
11486     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11487     in~a~'custom~line'.~They~are~incompatible.~
11488     The~key~'multiplicity'~will~be~ignored.
11489 }

11490 \@@_msg_new:nn { empty~environment }
11491 {
11492     Empty~environment.\\
11493     Your~ \@@_full_name_env: \ is~empty.
11494 }

```

```

11495 \@@_msg_new:nn { No~letter~and~no~command }
11496 {
11497   Erroneous~use.\\
11498   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
11499   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11500   '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11501 }
11502 \@@_msg_new:nn { Forbidden~letter }
11503 {
11504   Forbidden~letter.\\
11505   You~can't~use~the~letter~'#1'~for~a~customized~line.~
11506   It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11507 }
11508 \@@_msg_new:nn { Several~letters }
11509 {
11510   Wrong~name.\\
11511   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11512   have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11513 }
11514 \@@_msg_new:nn { Delimiter~with~small }
11515 {
11516   Delimiter~forbidden.\\
11517   You~can't~put~a~delimiter~in~the~preamble~of~your~
11518   \@@_full_name_env: \
11519   because~the~key~'small'~is~in~force.
11520 }
11521 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11522 {
11523   Unknown~cell.\\
11524   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11525   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11526   can't~be~executed~because~a~cell~doesn't~exist.~
11527   Your~command~ \token_to_str:N \line \ will~be~ignored.
11528 }
11529 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11530 {
11531   Duplicate~name. \\
11532   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11533   in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11534   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11535   { For~a~list~of~the~names~already~used,~type~H<return>. }
11536 }
11537 {
11538   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11539   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11540 }
11541 \@@_msg_new:nn { r~or~l~with~preamble }
11542 {
11543   Erroneous~use. \\
11544   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11545   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11546   your~ \@@_full_name_env: .~
11547   Your~key~will~be~ignored.
11548 }
11549 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11550 {
11551   Erroneous~use. \\
11552   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11553   in~an~exterior~column~of~the~array.
11554 }

```

```

11555 \@@_msg_new:nn { bad-corner }
11556 {
11557   Bad-corner. \\\
11558   '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11559   'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11560   This~specification~of~corner~will~be~ignored.
11561 }
11562 \@@_msg_new:nn { bad-border }
11563 {
11564   Bad-border. \\\
11565   '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11566   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11567   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11568   also~use~the~key~'tikz'
11569   \IfPackageLoadedF { tikz }
11570   { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11571   This~specification~of~border~will~be~ignored.
11572 }
11573 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11574 {
11575   TikZ~not~loaded. \\\
11576   You~can't~use~ \token_to_str:N \TikzEveryCell \
11577   because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11578   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11579   Your~command~will~be~ignored.
11580 }
11581 \@@_msg_new:nn { tikz~key~without~tikz }
11582 {
11583   TikZ~not~loaded. \\\
11584   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
11585   \Block '~because~you~have~not~loaded~TikZ.~
11586   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11587   before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11588 }
11589 \@@_msg_new:nn { Bad~argument~for~Block }
11590 {
11591   Bad~argument. \\\
11592   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11593   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11594   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
11595   the~argument~was~empty).
11596 }
11597 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
11598 {
11599   Erroneous~use. \\\
11600   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11601   'last-col'~without~value.~However,~you~can~go~on~for~this~time~
11602   (the~value~' \l_keys_value_tl '~will~be~ignored).
11603 }
11604 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
11605 {
11606   Erroneous~use. \\\
11607   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11608   'last-col'~without~value.~ However,~you~can~go~on~for~this~time~
11609   (the~value~' \l_keys_value_tl '~will~be~ignored).
11610 }
11611 \@@_msg_new:nn { Block~too~large~1 }
11612 {
11613   Block~too~large. \\\
11614   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
11615   too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.

```

```

11616 }
11617 \@@_msg_new:nn { Block-too-large-2 }
11618 {
11619   Block-too-large. \\
11620   The-preamble-of-your~ \@@_full_name_env:,~
11621   which-is~ '\exp_not:V\g_@@_user_preamble_tl',~announces~
11622   \int_use:N \g_@@_static_num_of_col_int \ columns~
11623   \@@_message_about_false_cols:
11624   but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
11625   specified-in-the-cell-#1-#2-can't-be-drawn.~
11626   \bool_if:NF \l_@@_light_syntax_bool
11627   {
11628     You-should-add-some-ampersands~(&)~at-the-end-of-the-first-row~
11629     of-your~\@@_full_name_env:~.~
11630   }
11631   This-block-will-be-ignored.
11632 }
11633 \@@_msg_new:nn { unknown-column-type }
11634 {
11635   Bad-column-type. \\
11636   The-column-type~'#1'~in-your~ \@@_full_name_env: \ is-unknown.
11637 }
11638 \@@_msg_new:nn { unknown-column-type-multicolumn }
11639 {
11640   Bad-column-type. \\
11641   The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
11642   ~of-your~ \@@_full_name_env: \ is-unknown.
11643 }
11644 \@@_msg_new:nn { unknown-column-type-S }
11645 {
11646   Bad-column-type. \\
11647   The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown.~
11648   If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should~
11649   load-that-package-with~\token_to_str:N \usepackage \{siunitx\}.
11650 }
11651 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
11652 {
11653   Bad-column-type. \\
11654   The-column-type~'S'~in-the-command~\token_to_str:N \multicolumn \
11655   of-your~ \@@_full_name_env: \ is-unknown.~If-you-want-to-use-the-column~
11656   type~'S'~of-siunitx,~you-should-load-that-package~(with~
11657   \token_to_str:N \usepackage \{siunitx\}).
11658 }
11659 \@@_msg_new:nn { unknown-column-type-; }
11660 {
11661   Bad-column-type. \\
11662   The-column-type~';'~in-your~ \@@_full_name_env: \ is-unknown.~
11663   You-should-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\})~
11664   in-order-to-use-it-for-vertical-dashed-rules.
11665 }
11666 \@@_msg_new:nn { unknown-column-type-RCL }
11667 {
11668   Bad-column-type. \\
11669   The-column-type~'#1'~in-your~ \@@_full_name_env: \ is-unknown.~
11670   Maybe-you-want-to-use~\str_lowercase:n { #1 }.
11671 }
11672 \@@_msg_new:nn { tabularnote-forbidden }
11673 {
11674   Forbidden-command. \\
11675   You-can't-use-the-command~ \token_to_str:N \tabularnote \

```

```

11676 ~here.~This~command~is~available~only~in~
11677 \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11678 the~argument~of~a~command~\token_to_str:N \caption \ included~
11679 in~an~environment~\{table\}.~Your~command~will~be~ignored.
11680 }
11681 \@@_msg_new:nn { borders~forbidden }
11682 {
11683   Forbidden~key.\\
11684   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11685   because~the~option~'rounded-corners'~is~in~force~with~a~non-zero~value.~
11686   Your~key~will~be~ignored.
11687 }
11688 \@@_msg_new:nn { bottomrule~without~booktabs }
11689 {
11690   booktabs~not~loaded.\\
11691   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11692   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11693   after~'nicematrix'.~Your~key~will~be~ignored.
11694 }
11695 \@@_msg_new:nn { enumitem~not~loaded }
11696 {
11697   enumitem~not~loaded. \\
11698   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11699   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11700   (before~or~after~'nicematrix').~All~the~commands~
11701   \token_to_str:N \tabularnote \ will~be~ignored~in~the~document.
11702 }
11703 \@@_msg_new:nn { tikz~without~tikz }
11704 {
11705   TikZ~not~loaded. \\
11706   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11707   should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11708   If~you~go~on,~your~key~will~be~ignored.
11709 }
11710 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11711 {
11712   TikZ~not~loaded. \\
11713   You~have~used~the~key~'tikz'~(or~'dashed')~in~the~definition~of~a~
11714   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
11715   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11716   use~that~custom~line.~You~should~load~TikZ~(with~
11717   \token_to_str:N \usepackage \{tikz\}).
11718 }
11719 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11720 {
11721   TikZ~not~loaded. \\
11722   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11723   command~ \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11724   Your~key~will~be~ignored.~You~should~load~TikZ~(with~
11725   \token_to_str:N \usepackage \{tikz\}).
11726 }
11727 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11728 {
11729   Erroneous~use.\\
11730   In~a~'custom~line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11731   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11732   The~key~'color'~will~be~ignored.
11733 }
11734 \@@_msg_new:nn { Wrong~last~row }
11735 {

```

```

11736 Wrong~number.\\
11737 You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
11738 \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11739 If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11740 last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11741 problem~by~using~'last-row'~without~value~(more~compilations~
11742 might~be~necessary).
11743 }
11744 \@@_msg_new:nn { Yet~in~env }
11745 {
11746   Nested~environments.\\
11747   Environments~of~nicematrix~can't~be~nested.~However~you~
11748   can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11749   or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11750   an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11751   that~box~in~another~environment~of~nicematrix.
11752 }
11753 \@@_msg_new:nn { Outside~math~mode }
11754 {
11755   Outside~math~mode.\\
11756   The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11757   (and~not~in~ \token_to_str:N \vcenter ).
11758 }
11759 \@@_msg_new:nn { One~letter~allowed }
11760 {
11761   Bad~name.\\
11762   The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11763   you~have~used~' \l_keys_value_tl '.~Your~key~will~be~ignored.
11764 }
11765 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11766 {
11767   Environment~\{TabularNote\}~forbidden.\\
11768   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11769   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11770   \{TabularNote\}~will~be~ignored.
11771 }
11772 \@@_msg_new:nn { varwidth~not~loaded }
11773 {
11774   varwidth~not~loaded.\\
11775   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11776   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11777   Your~column~will~behave~like~'p'.
11778 }
11779 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11780 {
11781   varwidth~not~loaded.\\
11782   You~can't~use~the~key~'V'~in~your~column~'X'~
11783   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11784   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11785 }
11786 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11787 {
11788   Unknown~key.\\
11789   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11790   \c_@@_available_keys_str
11791 }
11792 {
11793   The~available~keys~are:~color,~multiplicity,~sep-color,~tikz~
11794   and~total-width~(the~latter~is~meaningful~only~in~cunjunction~with~'tikz').
11795 }

```

If fact, there is also the key dotted but it won't be very useful since we provide \hdottedline, \cdottedline and the letter :.

```

11796 \@@_msg_new:nnn { Unknown~key~for~Block }
11797 {
11798   Unknown~key. \\
11799   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11800   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11801   \c_@@_available_keys_str
11802 }
11803 {
11804   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11805   b,~B,~borders,~c,~draw,~empty,~fill,~hlines,~hvlines,~l,~name,~
11806   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/color,~rules/width,
11807   ~t,~T,~tikz,~transparent~and~vlines.
11808 }
11809 \@@_msg_new:nnn { Unknown~key~for~Brace }
11810 {
11811   Unknown~key. \\
11812   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11813   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11814   Your~key~will~be~ignored. \\
11815   \c_@@_available_keys_str
11816 }
11817 {
11818   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11819   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11820   right~shorten)~and~yshift.
11821 }
11822 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11823 {
11824   Unknown~key. \\
11825   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11826   \c_@@_available_keys_str
11827 }
11828 {
11829   The~available~keys~are~(in~alphabetic~order):~
11830   delimiters/color,~
11831   rules~(with~the~subkeys~'color'~and~'width'),~
11832   sub~matrix~(several~subkeys)~
11833   and~xdots~(several~subkeys).~
11834   The~latter~is~for~the~command~ \token_to_str:N \line .
11835 }
11836 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11837 {
11838   Unknown~key. \\
11839   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11840   \c_@@_available_keys_str
11841 }
11842 {
11843   The~available~keys~are~(in~alphabetic~order):~
11844   create~cell~nodes,~
11845   delimiters/color~and~
11846   sub~matrix~(several~subkeys).
11847 }
11848 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11849 {
11850   Unknown~key. \\
11851   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11852   \c_@@_available_keys_str
11853 }
11854 {

```

```

11855     The~available~keys~are~(in~alphabetic~order):~
11856     'delimiters/color',~
11857     'extra-height',~
11858     'hlines',~
11859     'hvlines',~
11860     'left-xshift',~
11861     'name',~
11862     'right-xshift',~
11863     'rules'~(with~the~subkeys~'color'~and~'width'),~
11864     'slim',~
11865     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11866 }
11867 \@@_msg_new:nnn { Unknown~key~for~notes }
11868 {
11869     Unknown~key.\\
11870     The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11871     \c_@@_available_keys_str
11872 }
11873 {
11874     The~available~keys~are~(in~alphabetic~order):~
11875     bottomrule,~
11876     code~after,~
11877     code~before(+),~
11878     detect~duplicates,~
11879     enumitem~keys,~
11880     enumitem~keys~para,~
11881     para,~
11882     label~in~list,~
11883     label~in~tabular~and~
11884     style.
11885 }
11886 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11887 {
11888     Unknown~key.\\
11889     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11890     \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11891     \c_@@_available_keys_str
11892 }
11893 {
11894     The~available~keys~are~(in~alphabetic~order):~
11895     bold,~
11896     cell~space~top~limit(+),~
11897     cell~space~bottom~limit(+),~
11898     cell~space~limits(+),~
11899     color,~
11900     fill~(alias:~rowcolor),~
11901     nb~rows,~
11902     opacity~and~
11903     rounded~corners.
11904 }
11905 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11906 {
11907     Unknown~key.\\
11908     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11909     \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11910     \c_@@_available_keys_str
11911 }
11912 {
11913     The~available~keys~are~(in~alphabetic~order):~
11914     &~in~blocks,~
11915     allow~duplicate~names,~
11916     ampersand~in~blocks,~
11917     caption~above,~

```

```

11918 cell-space-bottom-limit(+),~
11919 cell-space-limits(+),~
11920 cell-space-top-limit(+),~
11921 color-of-false,~
11922 code-for-first-col(+),~
11923 code-for-first-row(+),~
11924 code-for-last-col(+),~
11925 code-for-last-row(+),~
11926 corners,~
11927 custom-key,~
11928 create-extra-nodes,~
11929 create-medium-nodes,~
11930 create-large-nodes,~
11931 custom-line,~
11932 delimiters~(several~subkeys),~
11933 end-of-row,~
11934 first-col,~
11935 first-row,~
11936 h(v)lines,~
11937 h(v)lines-except-borders,~
11938 last-col,~
11939 last-row,~
11940 left-margin,~
11941 light-syntax,~
11942 light-syntax-expanded,~
11943 matrix/columns-type,~
11944 no-cell-nodes,~
11945 notes~(several~subkeys),~
11946 nullify-dots,~
11947 pgf-node-code,~
11948 renew-dots,~
11949 renew-matrix,~
11950 respect-arraystretch,~
11951 rounded-corners,~
11952 right-margin,~
11953 rules~(with~the~subkeys~'color'~and~'width'),~
11954 small,~
11955 sub-matrix~(several~subkeys),~
11956 vl原因,~
11957 width-of-false(+),~
11958 xdots~(several~subkeys).
11959 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11960 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11961 {
11962   Unknown~key.\\
11963   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11964   \{NiceArray\}.~Your~key~will~be~ignored. \\
11965   \c_@@_available_keys_str
11966 }
11967 {
11968   The~available~keys~are~(in~alphabetic~order):~
11969   &-in-blocks,~
11970   ampersand-in-blocks,~
11971   b,~
11972   baseline,~
11973   c,~
11974   cell-space-bottom-limit,~
11975   cell-space-limits,~
11976   cell-space-top-limit,~
11977   code-after,~
11978   code-for-first-col(+),~

```

```

11979 code-for-first-row(+),~
11980 code-for-last-col(+),~
11981 code-for-last-row(+),~
11982 color-of-false,~
11983 columns-width,~
11984 corners,~
11985 create-blocks-in-col,~
11986 create-extra-nodes,~
11987 create-medium-nodes,~
11988 create-large-nodes,~
11989 draw-trees-in-col,~
11990 extra-left-margin,~
11991 extra-right-margin,~
11992 first-col,~
11993 first-row,~
11994 h(v)lines,~
11995 h(v)lines-except-borders,~
11996 last-col,~
11997 last-row,~
11998 left-margin,~
11999 light-syntax,~
12000 light-syntax-expanded,~
12001 name,~
12002 no-cell-nodes,~
12003 nullify-dots,~
12004 pgf-node-code,~
12005 renew-dots,~
12006 respect-arraystretch,~
12007 right-margin,~
12008 rounded-corners,~
12009 rules~(with~the~subkeys~'color'~and~'width'),~
12010 small,~
12011 t,~
12012 vlines,~
12013 width-of-false(+),~
12014 xdots/color,~
12015 xdots/shorten-start(+),~
12016 xdots/shorten-end(+),~
12017 xdots/shorten(+)-and~
12018 xdots/line-style.
12019 }

```

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`).

```

12020 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
12021 {
12022   Unknown~key.\\
12023   The~key~' \l_keys_key_str '~is~unknown~for~the~
12024   \@@_full_name_env: .~Your~key~will~be~ignored. \\
12025   \c_@@_available_keys_str
12026 }
12027 {
12028   The~available~keys~are~(in~alphabetic~order):~
12029   &-in-blocks,~
12030   ampersand-in-blocks,~
12031   b,~
12032   baseline,~
12033   c,~
12034   cell-space-bottom-limit,~
12035   cell-space-limits,~
12036   cell-space-top-limit,~
12037   code-after,~
12038   code-for-first-col(+),~

```

```

12039 code-for-first-row(+),~
12040 code-for-last-col(+),~
12041 code-for-last-row(+),~
12042 color-of-false,~
12043 columns-type,~
12044 columns-width,~
12045 corners,~
12046 create-blocks-in-col,~
12047 create-extra-nodes,~
12048 create-medium-nodes,~
12049 create-large-nodes,~
12050 draw-trees-in-col,~
12051 extra-left-margin,~
12052 extra-right-margin,~
12053 first-col,~
12054 first-row,~
12055 h(v)lines,~
12056 h(v)lines-except-borders,~
12057 l,~
12058 last-col,~
12059 last-row,~
12060 left-margin,~
12061 light-syntax,~
12062 light-syntax-expanded,~
12063 name,~
12064 no-cell-nodes,~
12065 nullify-dots,~
12066 pgf-node-code,~
12067 r,~
12068 renew-dots,~
12069 respect-arraystretch,~
12070 right-margin,~
12071 rounded-corners,~
12072 rules~(with~the~subkeys~'color'~and~'width'),~
12073 small,~
12074 t,~
12075 vlines,~
12076 width-of-false(+),~
12077 xdots/color,~
12078 xdots/shorten-start(+),~
12079 xdots/shorten-end(+),~
12080 xdots/shorten(+)-and~
12081 xdots/line-style.
12082 }

12083 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
12084 {
12085   Unknown~key.\\
12086   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
12087   \{NiceTabular\}.~Your~key~will~be~ignored. \\
12088   \c_@@_available_keys_str
12089 }
12090 {
12091   The~available~keys~are~(in~alphabetic~order):~
12092   &-in-blocks,~
12093   ampersand-in-blocks,~
12094   b,~
12095   baseline,~
12096   c,~
12097   caption,~
12098   cell-space-bottom-limit,~
12099   cell-space-limits,~
12100   cell-space-top-limit,~
12101   code-after,~

```

```

12102 code-for-first-col(+),~
12103 code-for-first-row(+),~
12104 code-for-last-col(+),~
12105 code-for-last-row(+),~
12106 color-of-false,~
12107 columns-width,~
12108 corners,~
12109 custom-line,~
12110 create-blocks-in-col,~
12111 create-extra-nodes,~
12112 create-medium-nodes,~
12113 create-large-nodes,~
12114 draw-trees-in-col,~
12115 extra-left-margin,~
12116 extra-right-margin,~
12117 first-col,~
12118 first-row,~
12119 h(v)lines,~
12120 h(v)lines-except-borders,~
12121 label,~
12122 last-col,~
12123 last-row,~
12124 left-margin,~
12125 light-syntax,~
12126 light-syntax-expanded,~
12127 name,~
12128 no-cell-nodes,~
12129 notes~(several~subkeys),~
12130 nullify-dots,~
12131 pgf-node-code,~
12132 renew-dots,~
12133 respect-arraystretch,~
12134 right-margin,~
12135 rounded-corners,~
12136 rules~(with~the~subkeys~'color'~and~'width'),~
12137 short-caption,~
12138 t,~
12139 tabularnote,~
12140 vl原因,~
12141 width-of-false(+),~
12142 xdots/color,~
12143 xdots/shorten-start(+),~
12144 xdots/shorten-end(+),~
12145 xdots/shorten(+)-and~
12146 xdots/line-style.
12147 }

12148 \@@_msg_new:nnn { Duplicate-name }
12149 {
12150   Duplicate-name.\
12151   The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
12152   the-same-environment-name-twice.~You-can-go-on,~but,~
12153   maybe,~you-will-have-incorrect-results-especially~
12154   if~you-use~'columns-width=auto'.~If~you-don't-want-to-see~this~
12155   message~again,~use~the-key~'allow-duplicate-names'~in~
12156   ' \token_to_str:N \NiceMatrixOptions '~.\
12157   \bool_if:NF \g_@@_messages_for_Overleaf_bool
12158     { For~a~list-of~the~names~already~used,~type~H~<return>. }
12159 }
12160 {
12161   The-names~already-defined-in~this~document~are:~
12162   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
12163 }

12164 \@@_msg_new:nn { caption-above-in-env }

```

```

12165 {
12166     The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~
12167     Your-key-will-be-ignored.
12168 }
12169 \@@_msg_new:nn { show-cell-names }
12170 {
12171     There-is-no-key~'show-cell-names'~in~nicematrix.\\
12172     You-should-use-the-command~\token_to_str:N \ShowCellNames\
12173     in-the~\token_to_str:N \CodeBefore\ or-the~\token_to_str:N
12174     \CodeAfter.~Your-key-will-be-ignored.
12175 }
12176 \@@_msg_new:nn { Option~auto~for~columns-width }
12177 {
12178     Erroneous~use.\\
12179     You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
12180     Your-key-will-be-ignored.
12181 }
12182 \@@_msg_new:nn { NiceTabularX~without~X }
12183 {
12184     NiceTabularX~without~X.\\
12185     You-should-not-use~\{NiceTabularX\}~without~X~columns.~However,~you-can-go-on.
12186 }
12187 \@@_msg_new:nn { Preamble~forgotten }
12188 {
12189     Preamble~forgotten.\\
12190     You-have-probably~forgotten-the-preamble-of-your~
12191     \@@_full_name_env: .
12192 }
12193 \@@_msg_new:nn { Invalid~col~number }
12194 {
12195     Invalid~column~number.\\
12196     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
12197     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
12198     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12199 }
12200 \@@_msg_new:nn { Invalid~row~number }
12201 {
12202     Invalid~row~number.\\
12203     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
12204     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
12205     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12206 }
12207 \@@_define_com:NNN p ( )
12208 \@@_define_com:NNN b [ ]
12209 \@@_define_com:NNN v | |
12210 \@@_define_com:NNN V \l \l
12211 \@@_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>	21
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	38
9	The <code>\CodeBefore</code>	54
10	The environment <code>{NiceArrayWithDelims}</code>	58
11	Construction of the preamble of the array	63
12	The redefinition of <code>\multicolumn</code>	82
13	The environment <code>{NiceMatrix}</code> and its variants	100
	13.1 Definition of <code>{pNiceMatrix}</code>	100
	13.2 The key <code>renew-matrix</code>	101
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	101
15	After the construction of the array	102
16	We draw the dotted lines	112
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	129
18	User commands available in the new environments	135
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	141
20	The command <code>\RowStyle</code>	143
21	Colors of cells, rows and columns	146
22	The vertical and horizontal rules	158
23	The empty corners	181
24	The environment <code>{NiceMatrixBlock}</code>	185
25	The extra nodes	186
26	The blocks	191
27	Automatic arrays	219
28	The redefinition of the command <code>\dotfill</code>	220
29	The command <code>\diagbox</code>	221

30	The keyword <code>\CodeAfter</code>	222
31	The delimiters in the preamble	223
32	The command <code>\SubMatrix</code>	224
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	233
34	The commands <code>HBrace</code> et <code>VBrace</code>	237
35	The command <code>TikzEveryCell</code>	239
36	The key <code>draw-trees-in-col</code>	241
37	The key <code>create-blocks-in-col</code>	243
38	The command <code>\ShowCellNames</code>	243
39	We process the options at package loading	245
40	About the package underscore	247
41	Compatibility with <code>threeparttable</code>	247
42	Gestion of the errors	247
	42.1 Security in the <code>CodeBefore</code> and the <code>CodeAfter</code>	247
	42.2 The error messages of the package	248