

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

September 5, 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 7.11c of `nicematrix`, at the date of 2026/09/05.

```

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[l]{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
```

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag (the `X` columns of `nicematrix` are inspired by those of `tabularx`). You will use that flag for the blocks.

```
359 \bool_new:N \l_@@_X_bool
```

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

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

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

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

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

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

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

`\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:`.

```
364 \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 }`).

```
365 \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).

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

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

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

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

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

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

```
370 \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`).

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

For `\multicolumn`.

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

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

```
373 \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`.

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

```
375 \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 `:`.

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

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

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

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.

```
386 \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”.

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

```
389 \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*.

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

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

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

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

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

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

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

```
395 \cs_new:Npn \@@_full_name_env:
396 {
397   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
398     { command \space \c_backslash_str \g_@@_name_env_str }
399     { environment \space \{ \g_@@_name_env_str \} }
400 }

401 \tl_new:N \g_@@_cell_after_hook_tl
```

The argument is given by curryfication.

```
402 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
403 { \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.

```
404 \tl_new:N \g_@@_pre_code_before_tl
405 \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`.

```
406 \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.

```
407 \tl_new:N \g_@@_pre_code_after_tl
408 \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.

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

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

```
410 \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.

```
411 \int_new:N \l_@@_old_iRow_int
412 \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).

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

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

```
414 \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.

```
415 \bool_new:N \l_@@_X_columns_aux_bool
416 \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.

```
417 \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).

```
418 \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).

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

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

```
421 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
433 \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”.

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

```
435 \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.

```
436 \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!).

```
437 \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.

```
438 \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: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
439 \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.

```
440 \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.

```
441 \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`).

```
442 \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.

```
443 \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).

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

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

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

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

```
447 \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.

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

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

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

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.

```

450 \dim_new:N \g_@@_delta_x_one_dim
451 \dim_new:N \g_@@_delta_y_one_dim
452 \dim_new:N \g_@@_delta_x_two_dim
453 \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).

```

454 \int_new:N \l_@@_row_min_int
455 \int_new:N \l_@@_row_max_int
456 \int_new:N \l_@@_col_min_int
457 \int_new:N \l_@@_col_max_int

458 \int_new:N \l_@@_initial_i_int
459 \int_new:N \l_@@_initial_j_int
460 \int_new:N \l_@@_final_i_int
461 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

462 \int_new:N \l_@@_segment_start_int
463 \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.

```

464 \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).

```

465 \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`.

```

466 \clist_new:N \g_@@_false_cols_clist
467 \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`.

```

468 \tl_new:N \l_@@_draw_tl
469 \seq_new:N \l_@@_tikz_seq
470 \tl_new:N \l_@@_tikz_rule_tl

```

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```

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

475 \bool_new:N \l_@@_fix_vertex_bool

```

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

```
476 \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).

```
477 \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.

```
478 \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}`).

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

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

```
480 \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`).

```
481 \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.

```
482 \int_new:N \l_@@_first_row_int
483 \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.

```
484 \int_new:N \l_@@_first_col_int
485 \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).

```
486 \int_new:N \l_@@_last_row_int
487 \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

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
489 \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 .

```
490 \int_new:N \l_@@_last_col_int
491 \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.

```
492 \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`).

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

Some utilities

```
494 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
495 {
```

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

```
496 \def \l_tmpa_tl { #1 }
497 \def \l_tmpb_tl { #2 }
498 }
```

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`.

```
499 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
500 {
501 \clist_if_in:NnF #1 { all }
502 {
503 \clist_clear:N \l_tmpa_clist
504 \clist_map_inline:Nn #1
505 {
506 \tl_if_head_eq_meaning:nNTF { ##1 } -
507 {
```

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.

```

508         \int_if_zero:nF { #2 }
509         {
510             \clist_put_right:Ne \l_tmpa_clist
511             { \int_eval:n { #2 + (##1) + 1 } }
512         }
513     }
514     {

```

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

```

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

```

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

```

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

```

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.

```

528 \AtBeginDocument
529 {
530     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
531     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
532 }

```

5 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is before the `{tabular}`.
- It’s also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:

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

```
533 \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`.

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

536 \seq_new:N \g_@@_notes_seq
537 \seq_new:N \g_@@_notes_in_caption_seq
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
550 \NewDocumentCommand { \tabularnote } { o m }
551 { \@@_err_enumitem_not_loaded: }

552 \AddToHook { package / enumitem / after }
553 {
```

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

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

One must remind that we have allowed a `\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`.

```
573 \RenewDocumentCommand { \tabularnote } { o m }
574 {
575     \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capytype }
576     {
577         \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
578         { \@@_error:n { tabularnote~forbidden } }
579     }
```

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

```
580 \bool_if:NTF \l_@@_in_caption_bool
581 {
582     \tl_if_novalue:nTF { #1 }
583     { \@@_tabularnote_capt:n { #1 } }
584     { \@@_tabularnote_capt:n { \exp_not:n { #1 } } }
585 }
586 {
587     \tl_if_novalue:nTF { #1 }
588     { \@@_tabularnote:n { #1 } }
589     { \@@_tabularnote:n { \exp_not:n { #1 } } }
```

```

590         }
591         { #2 }
592     }
593 }
594 }
595 }
596 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
597 {
598     \@@_error_or_warning:n { enumitem~not~loaded }
599     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
600 }
601 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
602 { \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`.

```

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

```

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).

```

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

```

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`.

```

608     \int_zero:N \l_tmpb_int
609     \seq_map_indexed_inline:Nn \g_@@_notes_seq
610     {
611         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
612         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
613         {
614             \tl_if_novalue:nTF { #1 }
615             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
616             { \int_set:Nn \l_tmpa_int { ##1 } }
617             \seq_map_break:
618         }
619     }
620     \int_if_zero:nF \l_tmpa_int
621     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
622 }
623 \int_if_zero:nT \l_tmpa_int
624 {
625     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
626     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
627 }
628 \seq_put_right:Ne \l_@@_notes_labels_seq
629 {
630     \tl_if_novalue:nTF { #1 }
631     {
632         \@@_notes_format:n

```

```

633         {
634             \int_eval:n
635             {
636                 \int_if_zero:nTF \l_tmpa_int
637                 \c@tabularnote
638                 \l_tmpa_int
639             }
640         }
641     }
642     { #1 }
643 }
644 \peek_meaning:NF \tabularnote
645 {

```

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`.

```

646     \hbox_set:Nn \l_tmpa_box
647     {

```

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

```

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

```

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

```

651     \int_gdecr:N \c@tabularnote
652     \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.

```

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

```

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

```

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`.

```

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

```

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

```

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

```

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

```

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.

```

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

```

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!

```

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

699    \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
700    { \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.

```

701    \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
702    {
703      \begin { pgfscope }
704      \pgfset
705      {
706        inner~sep = \c_zero_dim ,
707        minimum~size = \c_zero_dim
708      }
709      \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
710      \pgfnode
711      { rectangle }
712      { center }
713      {
714        \vbox_to_ht:nn

```

```

715         { \dim_abs:n { #5 - #3 } }
716     {
717         \vfill
718         \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
719     }
720 }
721 { #1 }
722 { }
723 \end { pgfscope }
724 }

```

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.

```

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

```

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).

```

748 \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.

```

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

```

752 \dim_new:N \l_@@_xdots_shorten_start_dim

```

```

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

```

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.

```

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

```

762 \tl_new:N \l_@@_xdots_line_style_tl
763 \tl_const:Nn \c_@@_standard_tl { standard }
764 \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`.

```

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

```

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

```

767 \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).

```

768 \bool_new:N \l_@@_auto_columns_width_bool
769 \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!

```

770 \bool_new:N \g_@@_create_cell_nodes_bool

```

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

```

771 \str_new:N \l_@@_name_str

```

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

```

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

```

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

```

774 \bool_new:N \l_@@_except_borders_bool

```

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

```
775 \dim_new:N \l_@@_width_of_false_dim
776 \dim_set:Nn \l_@@_width_of_false_dim { 2 pt }
```

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

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

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

```
832 radius .value_required:n = true ,
```

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

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

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

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

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

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

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

```
861 \keys_define:nn { nicematrix / Global }
862 {
863   width-of-false .dim_set:N = \l_@@_width_of_false_dim ,
864   width-of-false .value_required:n = true ,
865   width-of-false+ .code:n =
866     \dim_add:Nn \l_@@_width_of_false_dim { #1 } ,
867   width-of-false+ .value_required:n = true ,
868   width-of-false~+ .meta:n = { width-of-false+ = #1 } ,
869   color-of-false .tl_set:N = \l_@@_color_of_false_tl ,
870   color-of-false .value_required:n = true ,
871   fix-vertex .value_forbidden:n = true ,
872   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
873   brace-shift+ .code:n =
874     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
875   brace-shift+ .value_required:n = true ,
876   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
877   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
878   show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
879   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
```

```

880   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
881   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
882   &-in-blocks .meta:n = ampersand-in-blocks ,
883   no-cell-nodes .choices:nn = { true , false }
884   {
885     \tl_if_eq:NnTF \l_keys_choice_tl { true }
886     {
887       \bool_set_true:N \l_@@_no_cell_nodes_bool
888       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_inactive:
889     }
890     {
891       \bool_set_false:N \l_@@_no_cell_nodes_bool
892       \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
893     }
894   } ,
895   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.

```

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

```

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

```

```

937 light-syntax-expanded .choices:nn = { true , false }
938 {
939     \tl_if_eq:NnTF \l_keys_value_tl { true }
940     {
941         \bool_set_true:N \l_@@_light_syntax_bool
942         \bool_set_true:N \l_@@_light_syntax_expanded_bool
943     }
944     {
945         \bool_set_false:N \l_@@_light_syntax_bool
946         \bool_set_false:N \l_@@_light_syntax_expanded_bool
947     }
948 } ,
949 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.

```

950 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
951 end-of-row .value_required:n = true ,
952 end-of-row .initial:n = ; ,
953
954 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
955 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
956 last-row .int_set:N = \l_@@_last_row_int ,
957 last-row .default:n = -1 ,
958
959 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
960 code-for-first-col .value_required:n = true ,
961 code-for-first-col+ .code:n =
962     { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
963 code-for-first-col+ .value_required:n = true ,
964 code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
965
966 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
967 code-for-last-col .value_required:n = true ,
968 code-for-last-col+ .code:n =
969     { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
970 code-for-last-col+ .value_required:n = true ,
971 code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
972
973 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
974 code-for-first-row .value_required:n = true ,
975 code-for-first-row+ .code:n =
976     { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
977 code-for-first-row+ .value_required:n = true ,
978 code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
979
980 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
981 code-for-last-row .value_required:n = true ,
982 code-for-last-row+ .code:n =
983     { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
984 code-for-last-row+ .value_required:n = true ,
985 code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
986
987 hlines .clist_set:N = \l_@@_hlines_clist ,
988 hlines .default:n = all ,
989 vlines .clist_set:N = \l_@@_vlines_clist ,
990 vlines .default:n = all ,
991
992 vlines-in-sub-matrix .code:n =
993 {
994     \tl_if_single_token:nTF { #1 }
995     {
996         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
997         { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

998         { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
999     }
1000     { \@@_error:n { One~letter~allowed } }
1001 },
1002 vl_lines-in-sub-matrix .value_required:n = true ,
1003 hv_lines .code:n =
1004 {
1005     \bool_set_true:N \l_@@_hv_lines_bool
1006     \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
1007     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1008 } ,
1009 hv_lines .value_forbidden:n = true ,
1010 hv_lines-except-borders .code:n =
1011 {
1012     \tl_set_eq:NN \l_@@_vl_lines_clist \c_@@_all_tl
1013     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1014     \bool_set_true:N \l_@@_hv_lines_bool
1015     \bool_set_true:N \l_@@_except_borders_bool
1016 } ,
1017 hl_lines-except-borders .value_forbidden:n = true ,
1018 hl_lines-except-borders .code:n =
1019 {
1020     \tl_set_eq:NN \l_@@_hl_lines_clist \c_@@_all_tl
1021     \bool_set_true:N \l_@@_hl_lines_bool
1022     \bool_set_true:N \l_@@_except_borders_bool
1023 } ,
1024 hl_lines-except-borders .value_forbidden:n = true ,
1025 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
1026 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.

```

1027 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
1028 renew-dots .value_forbidden:n = true ,
1029 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
1030 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
1031 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
1032 create-extra-nodes .meta:n =
1033 { create-medium-nodes , create-large-nodes } ,
1034 left-margin .dim_set:N = \l_@@_left_margin_dim ,
1035 left-margin .default:n = \arraycolsep ,
1036 right-margin .dim_set:N = \l_@@_right_margin_dim ,
1037 right-margin .default:n = \arraycolsep ,
1038 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
1039 margin .default:n = \arraycolsep ,
1040 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1041 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1042 extra-margin .meta:n =
1043 { extra-left-margin = #1 , extra-right-margin = #1 } ,
1044 extra-margin .value_required:n = true ,
1045 respect-arraystretch .code:n =
1046 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1047 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.

```

1048 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1049 pgf-node-code .value_required:n = true ,
1050 }

```

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

```

1051 \keys_define:nn { nicematrix / environments }
1052 {
1053   draw-trees-in-col .code:n =
1054     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1055   draw-trees-in-col .value_required:n = true ,
1056   create-blocks-in-col .code:n =
1057     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1058   create-blocks-in-col .value_required:n = true ,
1059   corners .clist_set:N = \l_@@_corners_clist ,
1060   corners .default:n = { NW , SW , NE , SE } ,
1061   code-before .code:n =
1062     {
1063       \tl_if_empty:nF { #1 }
1064       {
1065         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1066         \bool_set_true:N \l_@@_code_before_bool
1067       }
1068     } ,
1069   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}`.

```

1070   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1071   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1072   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).

```

1073   baseline .tl_set:N = \l_@@_baseline_tl ,
1074   baseline .value_required:n = true ,
1075   baseline .initial:n = c ,
1076   columns-width .code:n =
1077     \str_case:nnF { #1 }
1078     {
1079       { auto } { \bool_set_true:N \l_@@_auto_columns_width_bool }
1080       { row-height } { \bool_set_true:N \l_@@_row_columns_width_bool }
1081     }
1082     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1083   columns-width .value_required:n = true ,
1084   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.

```

1085     \legacy_if:nF { measuring@ }
1086     {
1087       \str_set:Ne \l_@@_name_str { #1 }
1088       \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1089       { \@@_err_duplicate_names:n { #1 } }
1090       { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1091     } ,
1092   name .value_required:n = true ,
1093   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1094   code-after .value_required:n = true ,
1095 }

1096 \cs_set:Npn \@@_err_duplicate_names:n #1
1097 { \@@_error:nn { Duplicate~name } { #1 } }

1098 \keys_define:nn { nicematrix / notes }
1099 {
1100   no-print .bool_set:N = \l_@@_notes_no_print_bool ,
1101   para .bool_set:N = \l_@@_notes_para_bool ,
1102   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1103   code-before .value_required:n = true ,

```

```

1104 code-before+ .code:n =
1105   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1106 code-before+ .value_required:n = true ,
1107 code-before~+ .code:n =
1108   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1109 code-before~+ .value_required:n = true ,
1110 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1111 code-after .value_required:n = true ,
1112 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1113 style .cs_set:Np = \@@_notes_style:n #1 ,
1114 style .value_required:n = true ,
1115 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1116 label-in-tabular .value_required:n = true ,
1117 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1118 label-in-list .value_required:n = true ,
1119 enumitem-keys .code:n =
1120   {
1121     \AtBeginDocument
1122     {
1123       \IfPackageLoadedT { enumitem }
1124       { \setlist* [ tabularnotes ] { #1 } }
1125     }
1126   } ,
1127 enumitem-keys .value_required:n = true ,
1128 enumitem-keys-para .code:n =
1129   {
1130     \AtBeginDocument
1131     {
1132       \IfPackageLoadedT { enumitem }
1133       { \setlist* [ tabularnotes* ] { #1 } }
1134     }
1135   } ,
1136 enumitem-keys-para .value_required:n = true ,
1137 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1138 unknown .code:n =
1139   \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1140 }

```

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.

```

1141 \keys_define:nn { nicematrix / delimiters }
1142 {
1143   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1144   color .tl_set:N = \l_@@_delimiters_color_tl ,
1145   color .value_required:n = true ,
1146 }

```

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

```

1147 \keys_define:nn { nicematrix }
1148 {
1149   NiceMatrixOptions .inherit:n =
1150     { nicematrix / Global } ,
1151   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1152   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1153   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1154   NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,

```

```

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

```

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

```

1194 \keys_define:nn { nicematrix / NiceMatrixOptions }
1195 {
1196     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1197     delimiters / color .value_required:n = true ,
1198     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1199     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1200     delimiters .value_required:n = true ,
1201     width .dim_set:N = \l_@@_width_dim ,
1202     last-col .code:n =
1203     \tl_if_empty:nF { #1 }
1204     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1205     \int_zero:N \l_@@_last_col_int ,
1206     small .bool_set:N = \l_@@_small_bool ,
1207     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.

```

1208     renew-matrix .code:n = \@@_renew_matrix: ,
1209     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}`.

```
1210     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.

```
1211     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)`.

```
1212     \str_if_eq:eeTF { #1 } { auto }
1213     { \@@_error:n { Option~auto~for~columns~width } }
1214     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
1215     allow-duplicate-names .code:n =
1216     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1217     allow-duplicate-names .value_forbidden:n = true ,
1218     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1219     notes .value_required:n = true ,
1220     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1221     sub-matrix .value_required:n = true ,
1222     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1223     matrix / columns-type .value_required:n = true ,
1224     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1225     unknown .code:n =
1226     \@@_unknown_key:nn
1227     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1228     { Unknown-key-for-NiceMatrixOptions }
1229 }
```

`\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.

```
1230 \NewDocumentCommand \NiceMatrixOptions { m }
1231 { \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.

```
1232 \keys_define:nn { nicematrix / NiceMatrix }
1233 {
1234     last-col .code:n = \tl_if_empty:nTF { #1 }
1235     {
1236         \bool_set_true:N \l_@@_last_col_without_value_bool
1237         \int_set:Nn \l_@@_last_col_int { -1 }
1238     }
1239     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1240     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1241     columns-type .value_required:n = true ,
1242     l .meta:n = { columns-type = l } ,
1243     r .meta:n = { columns-type = r } ,
1244     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1245     delimiters / color .value_required:n = true ,
1246     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1247     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1248     delimiters .value_required:n = true ,
1249     small .bool_set:N = \l_@@_small_bool ,
1250     small .value_forbidden:n = true ,
1251     unknown .code:n =
```

```

1252 \@@_unknown_key:nn
1253 { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1254 { Unknown-key-for-NiceMatrix }
1255 }

```

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

```

1256 \keys_define:nn { nicematrix / NiceArray }
1257 {

```

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.

```

1258 small .bool_set:N = \l_@@_small_bool ,
1259 small .value_forbidden:n = true ,
1260 last-col .code:n = \tl_if_empty:nF { #1 }
1261 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1262 \int_zero:N \l_@@_last_col_int ,
1263 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1264 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1265 unknown .code:n =
1266 \@@_unknown_key:nn
1267 { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1268 { Unknown-key-for-NiceArray }
1269 }

1270 \keys_define:nn { nicematrix / pNiceArray }
1271 {
1272 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1273 last-col .code:n = \tl_if_empty:nF { #1 }
1274 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1275 \int_zero:N \l_@@_last_col_int ,
1276 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1277 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1278 delimiters / color .value_required:n = true ,
1279 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1280 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1281 delimiters .value_required:n = true ,
1282 small .bool_set:N = \l_@@_small_bool ,
1283 small .value_forbidden:n = true ,
1284 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1285 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1286 unknown .code:n =
1287 \@@_unknown_key:nn
1288 { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1289 { Unknown-key-for-NiceMatrix }
1290 }

```

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

```

1291 \keys_define:nn { nicematrix / NiceTabular }
1292 {

```

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.

```

1293 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1294 \bool_set_true:N \l_@@_width_used_bool ,
1295 width .value_required:n = true ,
1296 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1297 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1298 tabularnote .value_required:n = true ,
1299 caption .tl_set:N = \l_@@_caption_tl ,
1300 caption .value_required:n = true ,

```

```

1301 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1302 short-caption .value_required:n = true ,
1303 label .tl_set:N = \l_@@_label_tl ,
1304 label .value_required:n = true ,
1305 last-col .code:n = \tl_if_empty:nF { #1 }
1306 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1307 \int_zero:N \l_@@_last_col_int ,
1308 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1309 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1310 unknown .code:n =
1311 \@@_unknown_key:nn
1312 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1313 { Unknown-key-for-NiceTabular }
1314 }

```

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
```

```

1315 \keys_define:nn { nicematrix / CodeAfter }
1316 {
1317 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1318 delimiters / color .value_required:n = true ,
1319 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1320 rules .value_required:n = true ,
1321 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1322 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1323 sub-matrix .value_required:n = true ,
1324 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1325 }

```

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}`).

```

1326 \cs_new_protected:Npn \@@_cell_begin:
1327 {

```

`\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.

```
1328 \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).

```
1329 \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.

```

1330 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
1331 \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:

```

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

```
1332 \int_gincr:N \c@jCol
```

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

Here is a version with the standard syntax of L3.

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

We will use a version a little more efficient.

```
1333 \if_int_compare:w \c@jCol = \c_one_int
1334 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1335 \@@_begin_of_row:
1336 \fi:
1337 \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:`.

```
1338 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1339 \@@_tuning_not_tabular_begin:
1340 \@@_tuning_exterior_rows:
1341 \g_@@_row_style_tl
1342 }
1343 \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.

```
1344 \cs_new_protected:Npn \@@_tuning_first_last_row:
1345 {
1346 \if_int_compare:w \c@iRow = \c_zero_int
1347 \if_int_compare:w \c@jCol > \c_zero_int
1348 \l_@@_code_for_first_row_tl
1349 \@@_tuning_key_small:
1350 \xglobal \colorlet { nicematrix-first-row } { . }
1351 \fi:
1352 \else:
1353 \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1354 \fi:
1355 }
```

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

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

We will use a version a little more efficient.

```

1356 \cs_new_protected:Npn \@@_tuning_last_row:
1357 {
1358   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1359     \l_@@_code_for_last_row_tl
1360     \@@_tuning_key_small:
1361     \xglobal \colorlet { nicematrix-last-row } { . }
1362   \fi:
1363 }

```

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

```

1364 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1365 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1366 {
1367   \m@th
1368   $ % $

```

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

```

1369   \@@_tuning_key_small:
1370 }
1371 \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.

```

1372 \cs_new_protected:Npn \@@_begin_of_row:
1373 {
1374   \int_gincr:N \c@iRow
1375   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1376   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1377   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1378   \pgfpicture
1379   \pgfrememberpicturepositiononpagetrue
1380   \pgfcoordinate
1381     { \@@_env: - row - \int_use:N \c@iRow - base }
1382     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1383   \str_if_empty:NF \l_@@_name_str
1384     {
1385       \pgfnodealias
1386         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1387         { \@@_env: - row - \int_use:N \c@iRow - base }
1388     }
1389   \endpgfpicture
1390 }

```

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.

```

1391 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1392 {
1393   \if_int_compare:w \c@iRow = \c_zero_int
1394     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1395     \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1396     \fi:
1397     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1398     \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1399     \fi:
1400   \else:
1401     \if_int_compare:w \c@iRow = \c_one_int
1402     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1403     \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1404     \fi:
1405     \fi:
1406   \fi:
1407 }

1408 \cs_new_protected:Npn \@@_rotate_cell_box:
1409 {
1410   \box_rotate:Nn \l_@@_cell_box
1411   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1412   \bool_if:NTF \g_@@_rotate_c_bool
1413   {
1414     \hbox_set:Nn \l_@@_cell_box
1415     { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1416   }
1417   {
1418     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1419     {
1420       \vbox_set_top:Nn \l_@@_cell_box
1421       {
1422         \vbox_to_zero:n { }
1423         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1424         \box_use:N \l_@@_cell_box
1425       }
1426     }
1427   }
1428   \bool_gset_false:N \g_@@_rotate_bool
1429   \bool_gset_false:N \g_@@_rotate_c_bool
1430   \bool_gset_false:N \g_@@_rotate_minus_bool
1431 }

```

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.

```

1432 \cs_set_protected:Npn \@@_adjust_size_box:
1433 {
1434     \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1435     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1436     \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1437     \fi:
1438     \global \g_@@_blocks_wd_dim = \c_zero_dim
1439     \fi:
1440     \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1441     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1442     \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1443     \fi:
1444     \global \g_@@_blocks_dp_dim = \c_zero_dim
1445     \fi:
1446     \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1447     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1448     \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1449     \fi:
1450     \global \g_@@_blocks_ht_dim = \c_zero_dim
1451     \fi:
1452 }
1453 \cs_new_protected:Npn \@@_cell_end:
1454 {

```

The following command is nullified in the tabulars.

```

1455     \@@_tuning_not_tabular_end:
1456     \hbox_set_end:
1457     \@@_cell_end_i:
1458 }

```

```

\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.

```

1459 \cs_new_protected:Npn \@@_cell_end_i:
1460 {

```

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.

```

1461   \g_@@_cell_after_hook_tl
1462   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1463   \@@_adjust_size_box:
1464   \box_set_ht:Nn \l_@@_cell_box
1465   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1466   \box_set_dp:Nn \l_@@_cell_box
1467   { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

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

```

1468   \@@_update_max_cell_width:

```

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

```

1469   \@@_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.

```

1470   \bool_if:NNTF \g_@@_empty_cell_bool
1471   { \box_use_drop:N \l_@@_cell_box }
1472   {
1473     \bool_if:NNTF \g_@@_not_empty_cell_bool
1474     \@@_print_node_cell:
1475     {
1476       \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1477       \@@_print_node_cell:
1478       \else:

```

```

1479         \box_use_drop:N \l_@@_cell_box
1480     \fi:
1481 }
1482 }
1483 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1484     \global \g_@@_col_total_int = \c@jCol
1485 \fi:
1486 \global \let \g_@@_empty_cell_bool \c_false_bool
1487 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1488 }

```

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:

```

1489 \cs_new_protected:Npn \@@_update_max_cell_width:
1490 {
1491     \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1492         \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1493     \fi:
1494 }

```

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`).

```

1495 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1496 {
1497     \@@_math_toggle:
1498     \hbox_set_end:
1499     \bool_if:NF \g_@@_rotate_bool
1500     {
1501         \hbox_set:Nn \l_@@_cell_box
1502         {
1503             \makebox [ \l_@@_col_width_dim ] [ s ]
1504             { \hbox_unpack_drop:N \l_@@_cell_box }
1505         }
1506     }
1507     \@@_cell_end_i:
1508 }

```

```

1509 \pgfset
1510 {
1511     nicematrix / cell-node /.style =
1512     {
1513         inner~sep = \c_zero_dim ,
1514         minimum~width = \c_zero_dim
1515     }
1516 }

```

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.

```

1517 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1518 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1519 {

```

```

1520 \use:c
1521 {
1522   __siunitx_table_align_
1523   \bool_if:NTF \l__siunitx_table_text_bool
1524   \l__siunitx_table_align_text_tl
1525   \l__siunitx_table_align_number_str
1526   :n
1527 }
1528 { #1 }
1529 }

```

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!

```

1530 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1531 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1532 {
1533   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1534   \hbox:n
1535   {
1536     \pgfsys@markposition
1537     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1538   }
1539   #1
1540   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1541   \hbox:n
1542   {
1543     \pgfsys@markposition
1544     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1545   }
1546 }

1547 \cs_new_protected:Npn \@@_print_node_cell:
1548 {
1549   \socket_use:nn { nicematrix / siunitx-wrap }
1550   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1551 }

```

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

```

1552 \cs_new_protected:Npn \@@_node_cell_active:
1553 {
1554   \pgfpicture
1555   \pgfsetbaseline \c_zero_dim
1556   \pgfrememberpicturepositiononpagetrue
1557   \pgfset { nicematrix / cell-node }
1558   \pgfnode
1559   { rectangle }
1560   { base }
1561   {

```

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).

```

1562   \sys_if_engine_xetex:T { \set@color }
1563   \box_use:N \l_@@_cell_box
1564 }
1565 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1566 { \l_@@_pgf_node_code_tl }
1567 \str_if_empty:NF \l_@@_name_str
1568 {
1569   \pgfnodealias

```

```

1570      { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1571      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1572    }
1573    \endpgfpicture
1574  }
1575  \cs_set_eq:NN \@@_node_cell: \@@_node_cell_active:
1576  \cs_new_protected:Npn \@@_node_cell_inactive:
1577    { \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).

```

1578 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1579 {
1580   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1581     { g_@@_ #2 _ lines _ tl }
1582   {
1583     \use:c { @@ _ draw _ #2 : nnn }
1584     { \int_use:N \c@iRow }
1585     { \int_use:N \c@jCol }
1586     { \exp_not:n { #3 } }
1587   }
1588 }

1589 \cs_new_protected:Npn \@@_array:n
1590 {
1591   \dim_set:Nn \col@sep
1592   {
1593     \bool_if:NTF \l_@@_row_columns_width_bool
1594       { \c_zero_dim }
1595       { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1596   }
1597   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1598     { \def \@halignto { } }
1599     { \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`.

```

1600   \@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).
1601   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1602   }
1603 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. We use here a `\cs_set_eq:cN` instead of a `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```
1604 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1605 \cs_new_protected:Npn \@@_create_row_node:
1606 {
1607   \int_compare:nNt \c@iRow > \g_@@_last_row_node_int
1608   {
1609     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1610     \@@_create_row_node_i:
1611   }
1612 }
1613 \cs_new_protected:Npn \@@_create_row_node_i:
1614 {
```

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

```
1615   \hbox
1616   {
1617     \bool_if:NT \l_@@_code_before_bool
1618     {
1619       \vtop
1620       {
1621         \skip_vertical:N 0.5\arrayrulewidth
1622         \pgfsys@markposition
1623         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1624         \skip_vertical:N -0.5\arrayrulewidth
1625       }
1626     }
1627     \pgfpicture
1628     \pgfrememberpicturepositiononpagetrue
1629     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1630     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1631     \str_if_empty:NF \l_@@_name_str
1632     {
1633       \pgfnodealias
1634       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1635       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1636     }
1637     \endpgfpicture
1638   }
1639 }
```

```
1640 \cs_new_protected:Npn \@@_in_everycr:
1641 {
1642   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1643   \tbl_update_cell_data_for_next_row:
1644   \int_gzero:N \c@jCol
1645   \bool_gset_false:N \g_@@_after_col_zero_bool
1646   \bool_if:NF \g_@@_row_of_col_done_bool
1647   {
1648     \@@_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).

```
1649   \clist_if_empty:NF \l_@@_hlines_clist
1650   {
1651     \str_if_eq:eeF \l_@@_hlines_clist { all }
1652     {
1653       \clist_if_in:NeT
1654       \l_@@_hlines_clist
```

```

1655         { \int_eval:n { \c@iRow + 1 } }
1656     }
1657 {

```

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.

```

1658         \int_compare:nNnT \c@iRow > { -1 }
1659     {
1660         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1661             { \hrule height \arrayrulewidth width \c_zero_dim }
1662     }
1663 }
1664 }
1665 }
1666 }

```

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

```

1667 \cs_set_protected:Npn \@@_renew_dots:
1668 {
1669     \cs_set_eq:NN \ldots \@@_Ldots:
1670     \cs_set_eq:NN \cdots \@@_Cdots:
1671     \cs_set_eq:NN \vdots \@@_Vdots:
1672     \cs_set_eq:NN \ddots \@@_Ddots:
1673     \cs_set_eq:NN \iddots \@@_Iddots:
1674     \cs_set_eq:NN \dots \@@_Ldots:
1675     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1676 }

```

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.

```

1677 \cs_new_protected:Npn \@@_patch_booktabs: { }
1678 \AddToHook { package / booktabs / after }
1679 {
1680     \cs_set_protected:Npn \@@_patch_booktabs:
1681         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1682 }

```

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`.

```

1683 \cs_new_protected:Npn \@@_some_initialization:
1684 {
1685     \@@_everycr:
1686     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1687     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1688     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1689     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1690     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1691     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1692 }

```

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

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

`\@@_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`.

```
1693 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1694 {
```

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

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

```
1695 \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.

```
1696 \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.

```
1697 \fp_gzero:N \g_@@_total_X_weight_fp
1698 \bool_gset_false:N \g_@@_V_of_X_bool

1699 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1700 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1701 \@@_patch_booktabs:
1702 \box_clear_new:N \l_@@_cell_box
1703 \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}`).

```
1704 \bool_if:NT \l_@@_small_bool
1705 {
1706     \def \arraystretch { 0.47 }
1707     \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1708 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1709 }
```

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!

```
1710 \bool_if:NT \g_@@_create_cell_nodes_bool
1711 {
1712     \tl_put_right:Nn \@@_begin_of_row:
1713     {
1714         \pgfsys@markposition
1715         { \@@_env: - row - \int_use:N \c@iRow - base }
1716     }
1717     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1718 }
```

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`.

```

1719   \def \ar@ialign
1720   {
1721     \tbl_init_cell_data_for_table:
1722     \@@_some_initialization:
1723     \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`.

```

1724     \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1725     \halign
1726   }

```

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).

```

1727   \cs_set_eq:NN \@@_old_ldots: \ldots
1728   \cs_set_eq:NN \@@_old_cdots: \cdots
1729   \cs_set_eq:NN \@@_old_vdots: \vdots
1730   \cs_set_eq:NN \@@_old_ddots: \ddots
1731   \cs_set_eq:NN \@@_old_iddots: \iddots
1732   \bool_if:NTF \l_@@_standard_cline_bool
1733   { \cs_set_eq:NN \cline \@@_standard_cline: }
1734   { \cs_set_eq:NN \cline \@@_cline: }
1735   \bool_if:NTF \l_@@_no_cell_nodes_bool
1736   { \@@_redefine_xdots_no_cell_nodes: }
1737   { \@@_redefine_xdots: }
1738   \cs_set_eq:NN \Hline \@@_Hline:
1739   \cs_set_eq:NN \Hspace \@@_Hspace:
1740   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1741   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1742   \cs_set_eq:NN \Block \@@_Block:
1743   \cs_set_eq:NN \rotate \@@_rotate:
1744   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1745   \cs_set_eq:NN \dotfill \@@_dotfill:
1746   \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1747   \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1748   \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1749   \cs_set_eq:NN \diagbox \@@_diagbox:nn
1750   \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1751   \cs_set_eq:NN \TopRule \@@_TopRule
1752   \cs_set_eq:NN \MidRule \@@_MidRule
1753   \cs_set_eq:NN \BottomRule \@@_BottomRule
1754   \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1755   \cs_set_eq:NN \Hbrace \@@_Hbrace
1756   \cs_set_eq:NN \Vbrace \@@_Vbrace
1757   \cs_set_eq:NN \hdashedline \@@_hdashedline:
1758   \cs_set_eq:NN \cdashedline \@@_cdashedline:n
1759   \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1760   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1761   \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1762   \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1763   \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1764   \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1765   \cs_set_eq:NN \FalseRow \@@_FalseRow
1766   \int_if_zero:nTF \l_@@_first_row_int
1767   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1768   {
1769     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int

```

```

1770      { \cs_gset_eq:NN \@_tuning_exterior_rows: \@_tuning_last_row: }
1771    }
1772    \bool_if:NT \l_@@_renew_dots_bool { \@_renew_dots: }

```

We redefine `\multicolumn`⁷.

```

1773    \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!).

```

1774    \tl_if_exist:NT \l_@@_note_in_caption_tl
1775    {
1776      \tl_if_empty:NF \l_@@_note_in_caption_tl
1777      {
1778        \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1779        \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1780      }
1781    }

```

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).

```

1782    \seq_gclear:N \g_@@_multicolumn_cells_seq
1783    \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`).

```

1784    \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).

```

1785    \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`).

```

1786    \int_gzero:N \g_@@_row_total_int

```

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

```

1787    \int_gzero:N \g_@@_col_total_int
1788    \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1789    \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.

```

1790    \tl_gclear_new:N \g_@@_Cdots_lines_tl
1791    \tl_gclear_new:N \g_@@_Ldots_lines_tl
1792    \tl_gclear_new:N \g_@@_Vdots_lines_tl
1793    \tl_gclear_new:N \g_@@_Ddots_lines_tl
1794    \tl_gclear_new:N \g_@@_Iddots_lines_tl
1795    \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1796    \tl_gclear:N \g_nicematrix_code_before_tl
1797    \tl_gclear:N \g_@@_pre_code_before_tl

```

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.

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]`).

```

1798 \dim_zero_new:N \l_@@_left_delim_dim
1799 \dim_zero_new:N \l_@@_right_delim_dim
1800 \bool_if:NTF \g_@@_delims_bool
1801 {

```

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

```

1802 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1803 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1804 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1805 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1806 }
1807 {
1808 \dim_gset:Nn \l_@@_left_delim_dim
1809 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1810 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1811 }
1812 }

```

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.

```

1813 \cs_new_protected:Npn \@@_pre_array:
1814 {
1815 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1816 \int_gzero_new:N \c@iRow
1817 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1818 \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).

```

1819 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1820 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1821 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1822 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1823 \bool_if:NT \g_@@_aux_found_bool
1824 {
1825 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1826 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1827 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1828 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1829 }

```

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).

```

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

```

```

1839      { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1840    }

```

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”.

```

1841    \int_compare:nNtT \l_@@_last_row_int > { -2 }
1842    {
1843      \tl_put_right:Nn \@@_update_for_first_and_last_row:
1844      {
1845        \dim_compare:nNtT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1846        { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1847        \dim_compare:nNtT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1848        { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1849      }
1850    }

1851    \seq_gclear:N \g_@@_cols_vlism_seq
1852    \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1853    \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1854    \@@_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).

```

1855    \hbox_set:Nw \l_@@_the_array_box
1856    \skip_horizontal:N \l_@@_left_margin_dim
1857    \skip_horizontal:N \l_@@_extra_left_margin_dim
1858    \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).

```

1859    \m@th
1860    $ % $
1861    \bool_if:NtF \l_@@_light_syntax_bool
1862    { \use:c { @@-light-syntax } }
1863    { \use:c { @@-normal-syntax } }
1864  }

```

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

```

1865    \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1866    {
1867      \tl_set:Nn \l_tmpa_tl { #1 }
1868      \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1869      { \@@_rescan_for_spanish:N \l_tmpa_tl }
1870      \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1871      \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...

```

1872    \@@_pre_array:
1873  }

```

```

1874 \cs_new_protected:Npn \@@_redefine_xdots:
1875 {
1876   \cs_set_eq:NN \Ldots \@@_Ldots:
1877   \cs_set_eq:NN \Cdots \@@_Cdots:
1878   \cs_set_eq:NN \Vdots \@@_Vdots:
1879   \cs_set_eq:NN \Ddots \@@_Ddots:
1880   \cs_set_eq:NN \Iddots \@@_Iddots:
1881 }
1882 \cs_new_protected:Npn \@@_redefine_xdots_no_cell_nodes:
1883 {
1884   \cs_set_protected:Npn \Ldots { \@@_error_xdots:N \Ldots }
1885   \cs_set_protected:Npn \Cdots { \@@_error_xdots:N \Cdots }
1886   \cs_set_protected:Npn \Vdots { \@@_error_xdots:N \Vdots }
1887   \cs_set_protected:Npn \Ddots { \@@_error_xdots:N \Ddots }
1888   \cs_set_protected:Npn \Iddots { \@@_error_xdots:N \Iddots }
1889 }
1890 \cs_new_protected:Npn \@@_error_xdots:N #1
1891 { \@@_error:nn { xdots~without~cell~nodes } { #1 } }

```

9 The \CodeBefore

```

1892 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1893 \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).

```

1894 \cs_new_protected:Npn \@@_pre_code_before:
1895 {

```

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.

```

1896 \pgfsys@markposition { \@@_env: - position }
1897 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1898 \pgfpicture
1899 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1900 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1901 {
1902   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1903   \pgfcoordinate { \@@_env: - row - ##1 }
1904   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1905 }

```

Now, the recreation of the `col` nodes.

```

1906 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1907 {
1908   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1909   \pgfcoordinate { \@@_env: - col - ##1 }
1910   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1911 }

```

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

```

1912 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1913 \endpgfpicture

```

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

```

1914 \@@_create_diag_nodes:

```

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

```

1915 \@@_create_blocks_nodes:
1916 \IfPackageLoadedT { tikz }
1917 {
1918   \tikzset
1919   {
1920     every-picture / .style =
1921     { overlay , name~prefix = \@@_env: - }
1922   }
1923 }
1924 \cs_set_eq:NN \cellcolor \@@_cellcolor
1925 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1926 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1927 \cs_set_eq:NN \rowcolor \@@_rowcolor
1928 \cs_set_eq:NN \rowcolors \@@_rowcolors
1929 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1930 \cs_set_eq:NN \arraycolor \@@_arraycolor
1931 \cs_set_eq:NN \columncolor \@@_columncolor
1932 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1933 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1934 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1935 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1936 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1937 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1938 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1939 }

```

```

1940 \cs_new_protected:Npn \@@_exec_code_before:
1941 {

```

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...

```

1942 \clist_map_inline:Nn \l_@@_corners_cells_clist
1943 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1944 \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`.

```

1945 \@@_add_to_colors_seq:nn { { nocolor } } { }
1946 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1947 \group_begin:
1948 \@@_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`.

```

1949 \if_mode_math:
1950 \@@_exec_code_before_i:
1951 \else:
1952   $ % $
1953   \@@_exec_code_before_i:
1954   $ % $
1955 \fi:
1956 \group_end:
1957 }

```

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`).

```

1958 \cs_new_protected:Npn \@@_exec_code_before_i:
1959 {
1960   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1961   { \@@_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`.

```
1962 \exp_last_unbraced:No \@@_CodeBefore_keys:
1963 \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).

```
1964 \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.

```
1965 \@@_actually_color:
1966 \l_@@_code_before_tl
1967 \q_stop
1968 }
```

```
1969 \keys_define:nn { nicematrix / CodeBefore }
1970 {
1971   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1972   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1973   sub-matrix .value_required:n = true ,
1974   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1975   delimiters / color .value_required:n = true ,
1976   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1977 }

1978 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1979 {
1980   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1981   \@@_CodeBefore:w
1982 }
```

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.

```
1983 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1984 {
1985   \bool_if:NTF \g_@@_aux_found_bool
1986   {
1987     \@@_pre_code_before:
1988     \legacy_if:nF { measuring@ } { #1 }
1989   }
```

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).

```
1990 {
1991   \pgfsys@markposition { \@@_env: - position }
1992   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1993   \pgfpicture
1994   \pgf@relevantforpicturesizefalse
1995   \endpgfpicture
```

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

```
1996 \pgfpicture
1997 \pgfrememberpicturepositiononpagetrue
1998 \endpgfpicture
```

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

```
1999 \pgfpicture
2000 \pgf@relevantforpicturesizefalse
```

```

2001         \pgfrememberpicturerepositiononpagetrue
2002     \endpgfpicture

```

The following picture corresponds \@@_actually_color:

```

2003     \pgfpicture
2004     \pgf@relevantforpicturesizefalse
2005     \endpgfpicture
2006 }
2007 }

```

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.

```

2008 \cs_new_protected:Npn \@@_recreate_cell_nodes:
2009 {
2010     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
2011     {
2012         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
2013         \pgfcoordinate { \@@_env: - row - ##1 - base }
2014         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2015         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
2016         {
2017             \cs_if_exist:cT
2018             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
2019             {
2020                 \pgfsys@getposition
2021                 { \@@_env: - ##1 - #####1 - NW }
2022                 \@@_node_position:
2023                 \pgfsys@getposition
2024                 { \@@_env: - ##1 - #####1 - SE }
2025                 \@@_node_position_i:
2026                 \@@_pgf_rect_node:nnn
2027                 { \@@_env: - ##1 - #####1 }
2028                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
2029                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
2030             }
2031         }
2032     }
2033     \@@_create_extra_nodes:
2034     \@@_create_aliases_last:
2035 }

```

```

2036 \cs_new_protected:Npn \@@_create_aliases_last:
2037 {
2038     \int_step_inline:nn \c@iRow
2039     {
2040         \pgfnodealias
2041         { \@@_env: - ##1 - last }
2042         { \@@_env: - ##1 - \int_use:N \c@jCol }
2043     }
2044     \int_step_inline:nn \c@jCol
2045     {
2046         \pgfnodealias
2047         { \@@_env: - last - ##1 }
2048         { \@@_env: - \int_use:N \c@iRow - ##1 }
2049     }
2050     \pgfnodealias
2051     { \@@_env: - last - last }
2052     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
2053 }

```

```

2054 \cs_new_protected:Npn \@@_create_blocks_nodes:

```

```

2055 {
2056   \pgfpicture
2057   \pgf@relevantforpicturesizefalse
2058   \pgfrememberpicturepositiononpagetrue
2059   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2060   { \@@_create_one_block_node:nnnnn ##1 }
2061   \endpgfpicture
2062 }

```

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

```

2063 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2064 {
2065   \tl_if_empty:nF { #5 }
2066   {
2067     \@@_qpoint:n { col - #2 }
2068     \dim_set_eq:NN \l_tmpa_dim \pgf@x
2069     \@@_qpoint:n { #1 }
2070     \dim_set_eq:NN \l_tmpb_dim \pgf@y
2071     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2072     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2073     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2074     \@@_pgf_rect_node:nnnnn
2075     { \@@_env: - #5 }
2076     { \dim_use:N \l_tmpa_dim }
2077     { \dim_use:N \l_tmpb_dim }
2078     { \dim_use:N \l_@@_tmpc_dim }
2079     { \dim_use:N \pgf@y }
2080   }
2081 }

```

10 The environment `{NiceArrayWithDelims}`

```

2082 \NewDocumentEnvironment { NiceArrayWithDelims }
2083 { m m 0 { } m ! 0 { } t \CodeBefore }
2084 {
2085   \@@_provide_pgfsyspdfmark:
2086   \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.

```

2087   \bgroup

2088   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2089   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2090   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2091   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2092   \int_gzero:N \g_@@_block_box_int
2093   \dim_gzero:N \g_@@_width_last_col_dim
2094   \dim_gzero:N \g_@@_width_first_col_dim
2095   \bool_gset_false:N \g_@@_row_of_col_done_bool
2096   \str_if_empty:NT \g_@@_name_env_str
2097   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2098   \bool_if:NTF \l_@@_tabular_bool
2099   \mode_leave_vertical:
2100   \@@_test_if_math_mode:

```

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).

```

2101 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2102 \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.

```

2103 \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.

```

2104 \cs_if_exist:NT \tikz@library@external@loaded
2105 {
2106   \tikzexternaldisable
2107   \cs_if_exist:NT \ifstandalone
2108     { \tikzset { external / optimize = false } }
2109 }

```

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

```

2110 \int_gincr:N \g_@@_env_int
2111 \bool_if:NF \l_@@_block_auto_columns_width_bool
2112 { \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.

```

2113 \seq_gclear:N \g_@@_blocks_seq
2114 \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`.

```

2115 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2116 \seq_gclear:N \g_@@_pos_of_xdots_seq
2117 \tl_gclear_new:N \g_@@_code_before_tl
2118 \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.

```

2119 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2120 {
2121   \bool_gset_true:N \g_@@_aux_found_bool
2122   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2123 }
2124 { \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.

```

2125 \tl_gclear:N \g_@@_aux_tl
2126 \tl_if_empty:NF \g_@@_code_before_tl
2127 {
2128   \bool_set_true:N \l_@@_code_before_bool
2129   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2130 }
2131 \tl_if_empty:NF \g_@@_pre_code_before_tl
2132 { \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`.

```

2133 \bool_if:NTF \g_@@_delims_bool
2134 { \keys_set:nn { nicematrix / pNiceArray } }
2135 { \keys_set:nn { nicematrix / NiceArray } }

```

⁹e.g. `\color[rgb]{0.5,0.5,0}`

```
2136 { #3 , #5 }
```

```
2137 \@@_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:.

```
2138 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2139 }
```

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

```
2140 {
2141   \bool_if:NTF \l_@@_light_syntax_bool
2142   { \use:c { end @@-light-syntax } }
2143   { \use:c { end @@-normal-syntax } }
2144   $ % $
2145   \skip_horizontal:N \l_@@_right_margin_dim
2146   \skip_horizontal:N \l_@@_extra_right_margin_dim
2147   \hbox_set_end:
2148   \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.

```
2149 \bool_if:NT \l_@@_width_used_bool
2150 {
2151   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2152   { \@@_error_or_warning:n { width~without~X~columns } }
2153 }
```

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 $\text{l_@@_X_columns_dim}$ multiplied by x .

```
2154 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2155 \@@_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).

```
2156 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2157 {
2158   \bool_if:NF \l_@@_last_row_without_value_bool
2159   {
2160     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2161     {
2162       \@@_error_or_warning:n { Wrong~last~row }
2163       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2164     }
2165   }
2166 }
```

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

```
2167 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2168 \bool_if:NTF \g_@@_last_col_found_bool
2169 { \int_gdecr:N \c@jCol }
2170 { }
```

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

```

2171 \int_compare:nNtT \l_@@_last_col_int > { -1 }
2172 { \@@_error:n { last~col~not~used } }
2173 }

```

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

```

2174 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2175 \int_compare:nNtT \l_@@_last_row_int > { -1 }
2176 { \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).

```

2177 \int_if_zero:nT \l_@@_first_col_int
2178 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2179 \bool_if:nTF { ! \g_@@_delims_bool }
2180 {
2181   \str_if_eq:eeTF c \l_@@_baseline_tl
2182   \@@_use_arraybox_with_notes_c:
2183   {
2184     \str_if_eq:eeTF b \l_@@_baseline_tl
2185     \@@_use_arraybox_with_notes_b:
2186     \@@_use_arraybox_with_notes:
2187   }
2188 }

```

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).

```

2189 {
2190   \int_if_zero:nTF \l_@@_first_row_int
2191   {
2192     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2193     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2194   }
2195   { \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

```

2196 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2197 {
2198   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2199   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2200 }
2201 { \dim_zero:N \l_tmpb_dim }
2202 \hbox_set:Nn \l_tmpa_box
2203 {
2204   \m@th
2205   $ % $
2206   \@@_color:o \l_@@_delimiters_color_tl
2207   \exp_after:wN \left \g_@@_left_delim_tl
2208   \vcenter
2209   {

```

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.

```

2210 \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2211 \hbox
2212 {
2213   % \bool_if:NtF \l_@@_tabular_bool
2214   % { \skip_horizontal:n { - \tabcolsep } }
2215   % { \skip_horizontal:n { - \arraycolsep } }

```

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).

```

2216         \skip_horizontal:n
2217         { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2218     \@@_use_arraybox_with_notes_c:
2219     \skip_horizontal:n
2220     { - \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
2221     % \bool_if:NTF \l_@@_tabular_bool
2222     %   { \skip_horizontal:n { - \tabcolsep } }
2223     %   { \skip_horizontal:n { - \arraycolsep } }
2224 }

```

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

```

2225     \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2226 }
2227 \exp_after:wN \right \g_@@_right_delim_tl
2228 $ % $
2229 }

```

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.

```

2230     \bool_if:NTF \l_@@_delimiters_max_width_bool
2231     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2232     \@@_put_box_in_flow:
2233 }

```

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).

```

2234     \bool_if:NT \g_@@_last_col_found_bool
2235     { \skip_horizontal:N \g_@@_width_last_col_dim }
2236     \@@_test_false_columns:
2237     \bool_if:NT \l_@@_preamble_bool
2238     {
2239         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2240         { \@@_err_columns_not_used: }
2241     }
2242     \@@_after_array:

```

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

```

2243     \egroup

```

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

```

2244     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2245     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2246     \iow_now:Ne \@mainaux
2247     {
2248         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2249         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2250         { \exp_not:o \g_@@_aux_tl }
2251     }
2252     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2253     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2254 }

```

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

```

2255 \cs_new_protected:Npn \@@_err_columns_not_used:
2256 {
2257     \@@_warning:n { columns~not~used }
2258     \cs_gset:Npn \@@_err_columns_not_used: { }
2259 }

```

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 .

```

2260 \cs_new_protected:Npn \@@_compute_width_X:
2261 {
2262   \tl_gput_right:Ne \g_@@_aux_tl
2263   {
2264     \bool_set_true:N \l_@@_X_columns_aux_bool
2265     \dim_set:Nn \l_@@_X_columns_dim
2266     {

```

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`

```

2267       \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2268       { \dim_use:N \l_@@_X_columns_dim }
2269       {
2270         \dim_compare:nNnTF
2271         {
2272           \dim_abs:n
2273           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2274         }
2275         <
2276         { 0.001 pt }
2277         { \dim_use:N \l_@@_X_columns_dim }
2278         {
2279           \dim_eval:n
2280           {
2281             \l_@@_X_columns_dim
2282             +
2283             \fp_to_dim:n
2284             {
2285               (
2286                 \dim_eval:n
2287                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2288               )
2289               / \fp_use:N \g_@@_total_X_weight_fp
2290             }
2291           }
2292         }
2293       }
2294     }
2295   }
2296 }

```

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`.

```

2297 \cs_new_protected:Npn \@@_transform_preamble:
2298 {

```

First, some tuning.

```

2299   \@@_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}`.

```
2300 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
```

Now, some post-treatment.

```
2301 \@@_transform_preamble_ii:
2302 }
2303 \cs_new_protected:Npn \@@_transform_preamble_i:
2304 {
```

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).

```
2305 \int_zero: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`).

```
2306 \seq_gclear:N \g_@@_cols_vlism_seq
2307 \clist_gclear:N \g_@@_false_cols_clist
2308 \clist_gclear:N \g_@@_false_rows_clist
```

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

```
2309 \int_zero:N \l_tmpa_int
2310 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2311 {
2312   \tl_set:Nn \l_@@_array_preamble_tl
2313     { ! { \skip_horizontal:N \arrayrulewidth } }
2314 }
2315 {
2316   \clist_if_in:NnT \l_@@_vlines_clist 1
2317   {
2318     \tl_set:Nn \l_@@_array_preamble_tl
2319       { ! { \skip_horizontal:N \arrayrulewidth } }
2320   }
2321 }
2322 }
```

```
2323 \cs_new_protected:Npn \@@_transform_preamble_ii:
2324 {
2325   \@@_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}`.

```
2326 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2327 {
2328   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2329   { \bool_gset_true:N \g_@@_delims_bool }
2330 }
2331 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2332 \int_if_zero:nTF \l_@@_first_col_int
2333 { \tl_put_left:No \l_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2334 {
2335   \bool_if:NF \g_@@_delims_bool
2336   {
2337     \bool_if:NF \l_@@_tabular_bool
2338     {
2339       \clist_if_empty:NT \l_@@_vlines_clist
2340       {
```

```

2341         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2342         { \tl_put_left:Nn \l_@@_array_preamble_tl { @ { } } }
2343     }
2344 }
2345 }
2346 }
2347 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2348 { \tl_put_right:No \l_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2349 {
2350     \bool_if:NF \g_@@_delims_bool
2351     {
2352         \bool_if:NF \l_@@_tabular_bool
2353         {
2354             \clist_if_empty:NT \l_@@_vlines_clist
2355             {
2356                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2357                 { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { } } }
2358             }
2359         }
2360     }
2361 }

```

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.

```

2362     \tag_if_active:F
2363     {

```

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

```

2364     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2365     {
2366         \tl_put_right:Nn \l_@@_array_preamble_tl
2367         { > { \@@_err_too_many_cols: } 1 }
2368     }
2369 }
2370 }

```

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.

```

2371 \cs_new_protected:Npn \@@_rec_preamble:n #1
2372 {

```

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

```

2373     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2374     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2375     {

```

Now, the columns defined by \newcolumntype of array.

```

2376     \cs_if_exist:cTF { NC @ find @ #1 }
2377     {
2378         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2379         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl

```

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.

```

2380     }
2381     {
2382         \str_case:nnF { #1 }
2383         {
2384             { S } { \@@_fatal:n { unknown-column-type~S } }
2385             { ; } { \@@_fatal:n { unknown-column-type~; } }
2386             { R } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2387             { C } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2388             { L } { \@@_fatal:nn { unknown-column-type~RCL } { #1 } }
2389         }
2390         { \@@_fatal:nn { unknown-column-type } { #1 } }
2391     }
2392 }
2393 }

2394 \cs_new_protected:cpn { @@_> : } #1 #2
2395 {
2396     \tl_put_right:Nn \l_@@_pre_cell_tl { > { #2 } }
2397     \@@_rec_preamble:n
2398 }

2399 \cs_new_protected:Npn \@@_use_pre_cell_tl:
2400 {
2401     \tl_put_right:No \l_@@_array_preamble_tl \l_@@_pre_cell_tl
2402     \tl_clear:N \l_@@_pre_cell_tl
2403 }

```

For c, l and r

```

2404 \cs_new_protected:Npn \@@_c: #1
2405 {
2406     \@@_use_pre_cell_tl:
2407     \tl_put_right:Nn \l_@@_array_preamble_tl
2408     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2409     \@@_rec_preamble_after_col:n
2410 }

2411 \cs_new_protected:Npn \@@_l: #1
2412 {
2413     \@@_use_pre_cell_tl:
2414     \tl_put_right:Nn \l_@@_array_preamble_tl
2415     {
2416         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2417         l
2418         < \@@_cell_end:
2419     }

```

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

```

2420     \@@_rec_preamble_after_col:n
2421 }

2422 \cs_new_protected:Npn \@@_r: #1
2423 {
2424     \@@_use_pre_cell_tl:
2425     \tl_put_right:Nn \l_@@_array_preamble_tl
2426     {
2427         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2428         r
2429         < \@@_cell_end:
2430     }

```

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

```

2431     \@@_rec_preamble_after_col:n
2432 }

```

For ! and @

```

2433 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2434 {
2435   \tl_put_right:Nn \l_@@_array_preamble_tl { #1 { #2 } }
2436   \@@_rec_preamble:n
2437 }
2438 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2439 \cs_new_protected:cpn { @@ _ | : } #1
2440 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2441   \int_incr:N \l_tmpa_int
2442   \@@_make_preamble_i_i:n
2443 }
2444 \cs_new_protected:Npn \@@_F: #1
2445 {
2446   \tl_put_right:Nn \l_@@_array_preamble_tl
2447   {
2448     > {
2449       \skip_horizontal:n { -2 \col@sep + \l_@@_width_of_false_dim }
2450       \skip_horizontal:N \c_zero_dim
2451       \@@_cell_begin:
2452     }
2453     c
2454     < {
2455       \@@_cell_end:
2456     }
2457   }

```

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

```

2458   \clist_gput_right:Ne \g_@@_false_cols_clist
2459   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2460   \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 <.

```

2461   \@@_rec_preamble_after_col:n
2462 }
2463 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2464 {

```

Here, we can’t use \str_if_eq:eeTF.

```

2465   \str_if_eq:nnTF { #1 } { | }
2466   { \use:c { @@ _ | : } | }
2467   { \@@_make_preamble_i_ii:nn { } #1 }
2468 }

```

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

```

2469 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2470 {
2471   \str_if_eq:nnTF { #2 } { [ ] }
2472   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2473   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2474 }
2475 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2476 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2477 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2478 {
2479   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2480   \tl_put_right:Ne \l_@@_array_preamble_tl
2481   {

```

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

```

2482     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2483   }
2484   \tl_gput_right:Ne \g_@@_rules_tl
2485   {

```

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:

```

2486   {
2487     \int_compare:nNnT \l_tmpa_int > 1
2488       { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2489     \int_set:Nn \l_@@_position_int
2490       { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2491     \dim_set:Nn \l_@@_rule_width_after_dim
2492       { \dim_use:N \l_@@_rule_width_before_dim }
2493     \@@_draw_vrule:n { #2 }
2494   }

```

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.

```

2495   }
2496   \int_zero:N \l_tmpa_int
2497   \str_if_eq:nnT { #1 } { \s_stop }
2498     { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }
2499   \@@_rec_preamble:n #1
2500 }

```

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.

```

2501 \keys_define:nn { nicematrix / p-column }
2502 {
2503   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2504   r .value_forbidden:n = true ,
2505   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2506   c .value_forbidden:n = true ,
2507   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2508   l .value_forbidden:n = true ,
2509   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2510   S .value_forbidden:n = true ,
2511   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2512   p .value_forbidden:n = true ,
2513   t .meta:n = p ,
2514   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2515   m .value_forbidden:n = true ,
2516   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2517   b .value_forbidden:n = true
2518 }

```

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

```

2519 \cs_new_protected:Npn \@@_p: #1
2520 {
2521   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2522 \@@_make_preamble_ii_i:n
2523 }
2524 \cs_new_eq:NN \@@_b: \@@_p:
2525 \cs_new_eq:NN \@@_m: \@@_p:
2526 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2527 {
2528   \str_if_eq:nnTF { #1 } { [ ] }
2529   { \@@_make_preamble_ii_ii:w [ ] }
2530   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2531 }
2532 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2533 { \@@_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.

```

2534 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2535 {

```

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).

```

2536   \str_set:Nn \l_@@_hpos_col_str { j }
2537   \@@_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`.

```

2538   \setlength { \l_tmpa_dim } { #2 }
2539   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2540 }
2541 \cs_new_protected:Npn \@@_keys_p_column:n #1
2542 { \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.

```

2543 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2544 {

```

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

```

2545   \use:e
2546   {
2547     \@@_make_preamble_ii_vi:nnnnnnnn
2548     { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2549     { #1 }
2550     {
2551       \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.

```

2552       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2553       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2554       {

```

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

```

2555         \def \exp_not:N \l_@@_hpos_cell_tl
2556         { \str_lowercase:f { \l_@@_hpos_col_str } }
2557     }
2558     \IfPackageLoadedTF { ragged2e }
2559     {
2560         \str_case:on \l_@@_hpos_col_str
2561         {

```

The following `\exp_not:N` are mandatory.

```

2562         c { \exp_not:N \Centering }
2563         l { \exp_not:N \RaggedRight }
2564         r { \exp_not:N \RaggedLeft }
2565     }
2566 }
2567 {
2568     \str_case:on \l_@@_hpos_col_str
2569     {
2570         c { \exp_not:N \centering }
2571         l { \exp_not:N \raggedright }
2572         r { \exp_not:N \raggedleft }
2573     }
2574 }
2575 #3
2576 }
2577 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2578 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2579 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2580 { #2 }
2581 {
2582     \str_case:onF \l_@@_hpos_col_str
2583     {
2584         { j } { c }
2585         { si } { c }
2586     }

```

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

```

2587         { \str_lowercase:f \l_@@_hpos_col_str }
2588     }
2589 }

```

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

```

2590     \@@_rec_preamble_after_col:n
2591 }

```

#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*.

```

2592 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2593 {
2594     % \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2595     % {
2596     %     \tl_put_right:Nn \l_@@_array_preamble_tl

```

```

2597 % { > \@@_test_if_empty_for_S: }
2598 % }
2599 % {
2600 % \str_if_eq:eeTF { #7 } { varwidth }
2601 % {
2602 % \tl_put_right:Nn \l_@@_array_preamble_tl
2603 % { > \@@_test_if_empty_varwidth: }
2604 % }
2605 % { \tl_put_right:Nn \l_@@_array_preamble_tl { > \@@_test_if_empty: } }
2606 % }
2607 \tl_put_right:Ne \l_@@_array_preamble_tl
2608 {
2609 >
2610 \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2611 { \@@_test_if_empty_for_S: }
2612 {
2613 \str_if_eq:eeTF { #7 } { varwidth }
2614 \@@_test_if_empty_varwidth:
2615 \@@_test_if_empty:
2616 }
2617 }
2618 \@@_use_pre_cell_tl:
2619 \tl_put_right:Nn \l_@@_array_preamble_tl
2620 {
2621 > {

```

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.

```

2622 \dim_set:Nn \l_@@_col_width_dim { #2 }
2623 \@@_cell_begin:

```

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

```

2624 \use:c { #7 } [ #1 ] { #2 }

```

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

```

2625 \everypar
2626 {
2627 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2628 \everypar { }
2629 }

```

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

```

2630 #3

```

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

```

2631 \g_@@_row_style_tl
2632 \arraybackslash
2633 #5
2634 }
2635 #8
2636 < {
2637 #6

```

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

```

2638 \@finalstrut \@arstrutbox
2639 \use:c { end #7 }

```

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

```

2640 #4
2641 \@@_cell_end:
2642 }
2643 }
2644 }

```

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

```
2645 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2646 {
```

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 `&`).

```
2647 \group_align_safe_begin:
2648 \peek_meaning:NTF &
2649 \@@_the_cell_is_empty: % contains \group_align_safe_end:
2650 {
2651 \peek_meaning:NTF \\\
2652 \@@_the_cell_is_empty: % idem
2653 {
2654 \peek_meaning:NTF \crcr
2655 \@@_the_cell_is_empty: % idem
2656 \group_align_safe_end:
2657 }
2658 }
2659 }
```

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

```
2660 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2661 {
2662 \group_align_safe_begin:
2663 \peek_meaning:NTF &
2664 \@@_the_cell_is_empty_varwidth:
2665 {
2666 \peek_meaning:NTF \\\
2667 \@@_the_cell_is_empty_varwidth:
2668 {
2669 \peek_meaning:NTF \crcr
2670 \@@_the_cell_is_empty_varwidth:
2671 \group_align_safe_end:
2672 }
2673 }
2674 }

2675 \cs_new_protected:Npn \@@_the_cell_is_empty:
2676 {
2677 \group_align_safe_end:
2678 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2679 {
```

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

```
2680 \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).

```
2681 \skip_horizontal:N \l_@@_col_width_dim
2682 }
2683 }

2684 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2685 {
2686 \group_align_safe_end:
2687 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2688 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2689 }
```

```

2690 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2691 {
2692   \peek_meaning:NT \__siunitx_table_skip:n
2693   { \bool_gset_true:N \g_@@_empty_cell_bool }
2694 }

```

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.

```

2695 \cs_new_protected:Npn \@@_center_cell_box:
2696 {

```

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

```

2697   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2698   {
2699     \dim_compare:nNnT
2700     { \box_ht:N \l_@@_cell_box }
2701     >

```

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).

```

2702     { \box_ht:N \strutbox }
2703     {
2704       \hbox_set:Nn \l_@@_cell_box
2705       {
2706         \box_move_down:nn
2707         {
2708           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2709             + \baselineskip ) / 2
2710         }
2711         { \box_use:N \l_@@_cell_box }
2712       }
2713     }
2714   }
2715 }

```

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

```

2716 \cs_new_protected:Npn \@@_V: #1 #2
2717 {
2718   \str_if_eq:nnTF { #2 } { [ ] }
2719   { \@@_make_preamble_V_i:w [ ] }
2720   { \@@_make_preamble_V_i:w [ ] { #2 } }
2721 }
2722 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2723 { \@@_make_preamble_V_ii:nn { #1 } }
2724 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2725 {
2726   \str_set:Nn \l_@@_vpos_col_str { p }
2727   \str_set:Nn \l_@@_hpos_col_str { j }
2728   \@@_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`.

```

2729   \setlength { \l_tmpa_dim } { #2 }
2730   \IfPackageLoadedTF { varwidth }
2731   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2732   {

```

```

2733     \@@_error_or_warning:n { varwidth-not-loaded }
2734     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2735 }
2736 }
2737 % \end{macrocod}
2738 %
2739 % \medskip
2740 % For |w| and |W|
2741 % \begin{macrocode}
2742 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2743 \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.

```

2744 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2745 {
2746   \tl_if_in:nnTF { clr } { #3 }
2747   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2748   {
2749     \@@_error:nn { Invalid~argument~for~w } { #3 }
2750     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2751   }
2752 }
2753 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2754 {
2755   \str_if_eq:nnTF { #3 } { s }
2756   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2757   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2758 }

```

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.

```

2759 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2760 {
2761   \@@_use_pre_cell_tl:
2762   \tl_put_right:Nn \l_@@_array_preamble_tl
2763   {
2764     > {

```

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

```

2765     \setlength { \l_@@_col_width_dim } { #2 }
2766     \@@_cell_begin:
2767     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2768   }
2769   c
2770   < {
2771     \@@_cell_end_for_w_s:
2772     #1
2773     \@@_adjust_size_box:
2774     % \box_use_drop:N \l_@@_cell_box
2775   }
2776 }

```

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

```

2777   \@@_rec_preamble_after_col:n
2778 }

```

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

```
2779 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2780 {
2781   \@@_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...

```
2782   \setlength { \l_tmpa_dim } { #4 }
2783   \tl_put_right:Ne \l_@@_array_preamble_tl
2784   {
2785     > {
```

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.

```
2786       \dim_set:Nn \l_@@_col_width_dim { \dim_use:N \l_tmpa_dim }
2787       \hbox_set:Nw \l_@@_cell_box
2788       \@@_cell_begin:
2789       \tl_set:Nn \exp_not:N \l_@@_hpos_cell_tl { #3 }
2790     }
2791   }
2792   \tl_put_right:Nn \l_@@_array_preamble_tl
2793   {
2794     c
2795     < {
2796       \@@_cell_end:
2797       \hbox_set_end:
2798       #1
2799       \@@_adjust_size_box:
2800       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2801     }
2802   }
```

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

```
2803   \@@_rec_preamble_after_col:n
2804 }

2805 \cs_new_protected:Npn \@@_special_W:
2806 {
2807   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2808   { \@@_warning:n { W-warning } }
2809 }
```

For `S` (of `siunitx`).

```
2810 \AddToHook { package / siunitx / after }
2811 {
2812   \cs_new_protected:Npn \@@_S: #1 #2
2813   {
2814     \str_if_eq:nnTF { #2 } { [ ] }
2815     { \@@_make_preamble_S:w [ ] }
2816     { \@@_make_preamble_S:w [ ] { #2 } }
2817   }
2818 }

2819 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2820 { \@@_make_preamble_S_i:n { #1 } }
```

```

2821 \cs_new_protected:Npn \@@_test_siunitx:
2822 {
2823   \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2824   { \@@_fatal:n { siunitx~too~old } }
2825   \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2826 }
2827 % \end{macrocode}
2828 %
2829 % \begin{macrocode}
2830 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2831 {
2832   \@@_test_siunitx:
2833   \@@_use_pre_cell_tl:
2834   \tl_put_right:Nn \l_@@_array_preamble_tl
2835   {
2836     > {

```

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).

```

2837       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2838       \keys_set:nn { siunitx } { #1 }
2839       \@@_cell_begin:
2840       \siunitx_cell_begin:w
2841     }
2842     c
2843     <
2844     {
2845       \siunitx_cell_end:

```

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

```

2846       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2847       {
2848         \bool_if:NTF \l__siunitx_table_text_bool
2849         \bool_set_true:N
2850         \bool_set_false:N
2851         \l__siunitx_table_text_bool
2852       }
2853       \@@_cell_end:
2854     }
2855   }

```

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

```

2856       \@@_rec_preamble_after_col:n
2857     }

```

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

```

2858 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2859 {
2860   \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.

```

2861   \int_if_zero:nTF \g_@@_static_num_of_col_int
2862   {
2863     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2864     {

```

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

```

2865       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2866       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl

```

```

2867         \@@_rec_preamble:n #2
2868     }
2869     {
2870         \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2871         \@@_make_preamble_iv:nn { #1 } { #2 }
2872     }
2873 }
2874 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2875 }
2876 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2877 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2878 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2879 {
2880     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2881     {
2882         \@@_delimiter:nnn #1
2883         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2884         \c_true_bool
2885     }
2886     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2887     {
2888         \@@_error:nn { delimiter~after~opening } { #2 }
2889         \@@_rec_preamble:n
2890     }
2891     { \@@_rec_preamble:n #2 }
2892 }

```

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

```

2893 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2894 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2895 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2896 {
2897     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2898     \tl_if_in:nnTF { ) ] \} } { #2 }
2899     { \@@_make_preamble_v:nnn #1 #2 }
2900     {
2901         \str_if_eq:nnTF \s_stop { #2 }
2902         {
2903             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2904             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2905             {
2906                 \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } }
2907                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2908                 {
2909                     \@@_delimiter:nnn #1
2910                     { \int_use:N \g_@@_static_num_of_col_int }
2911                     \c_false_bool
2912                 }
2913                 \@@_rec_preamble:n #2
2914             }
2915         }
2916         {
2917             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2918             { \tl_put_right:Nn \l_@@_array_preamble_tl { ! { \enskip } } }
2919             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2920             {
2921                 \@@_delimiter:nnn #1
2922                 { \int_use:N \g_@@_static_num_of_col_int }

```

```

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

2968 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2969 { \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 @{...}.

```

2970 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2971 {
2972     \int_gincr:N \g_@@_static_num_of_col_int
2973     \str_if_eq:nnTF { #1 } { < }
2974     { \@@_rec_preamble_after_col_i:n }
2975     {
2976         \str_if_eq:nnTF { #1 } { @ }
2977         { \@@_rec_preamble_after_col_ii:n }
2978         {
2979             \str_if_eq:eeTF \l_@@_vlines_clist { all }

```

```

2980         {
2981             \tl_put_right:Nn \l_@@_array_preamble_tl
2982             { ! { \skip_horizontal:N \arrayrulewidth } }
2983         }
2984         {
2985             \clist_if_in:NeT \l_@@_vlines_clist
2986             { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
2987             {
2988                 \tl_put_right:Nn \l_@@_array_preamble_tl
2989                 { ! { \skip_horizontal:N \arrayrulewidth } }
2990             }
2991         }
2992         \@@_rec_preamble:n { #1 }
2993     }
2994 }
2995 }
2996 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2997 {
2998     \tl_put_right:Nn \l_@@_array_preamble_tl { < { #1 } }
2999     \@@_rec_preamble_after_col:n
3000 }

```

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.

```

3001 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
3002 {
3003     \str_if_eq:eeTF \l_@@_vlines_clist { all }
3004     {
3005         \tl_put_right:Nn \l_@@_array_preamble_tl
3006         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3007     }
3008     {
3009         \clist_if_in:NeTF \l_@@_vlines_clist
3010         { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3011         {
3012             \tl_put_right:Nn \l_@@_array_preamble_tl
3013             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
3014         }
3015         { \tl_put_right:Nn \l_@@_array_preamble_tl { @ { #1 } } }
3016     }
3017     \@@_rec_preamble:n
3018 }

```

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.

```

3019 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
3020 { \@@_rec_preamble:n }
3021 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
3022 {
3023     \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.

```

3024     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 \NC@find } }
3025     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
3026 }

```

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.

```

3027 \cs_new_protected:Npn \@@_X: #1 #2

```

```

3028 {
3029   \str_if_eq:nnTF { #2 } { [ ] }
3030   { \@@_make_preamble_X:w [ ] }
3031   { \@@_make_preamble_X:w [ ] #2 }
3032 }
3033 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
3034 { \@@_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`.

```

3035 \keys_define:nn { nicematrix / X-column }
3036 {
3037   V .code:n =
3038     \IfPackageLoadedTF { varwidth }
3039     {
3040       \bool_set_true:N \l_@@_V_of_X_bool
3041       \bool_gset_true:N \g_@@_V_of_X_bool
3042     }
3043     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
3044   unknown .code:n =
3045     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
3046     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
3047     { \@@_error_or_warning:n { invalid~weight } }
3048 }

```

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

```

3049 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
3050 {

```

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).

```

3051   \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).

```

3052   \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`).

```

3053   \fp_set:Nn \l_tmpa_fp { 1.0 }
3054   \@@_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.

```

3055   \bool_set_false:N \l_@@_V_of_X_bool
3056   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

3057   \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).

```

3058   \bool_if:NTF \l_@@_X_columns_aux_bool
3059   {
3060     \@@_make_preamble_ii_iv:nnn

```

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

```

3061     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
3062     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } } }
3063     { \@@_no_update_width: }
3064   }

```

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.

```

3065 {
3066   \tl_put_right:Nn \l_@@_array_preamble_tl
3067   {
3068     > {
3069       \@@_cell_begin:
3070       \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`.

```

3071       \NotEmpty

```

The following code will nullify the box of the cell.

```

3072       \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3073       { \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`.

```

3074       \begin { minipage } { 5 cm } \arraybackslash
3075     }
3076     c
3077     < {
3078       \end { minipage }
3079       \@@_cell_end:
3080     }
3081   }
3082   \@@_rec_preamble_after_col:n
3083 }
3084 }

3085 \cs_new_protected:Npn \@@_no_update_width:
3086 {
3087   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
3088   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
3089 }

```

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

```

3090 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
3091 {
3092   \seq_gput_right:Ne \g_@@_cols_vlism_seq
3093   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
3094   \tl_put_right:Ne \l_@@_array_preamble_tl
3095   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
3096   \@@_rec_preamble:n
3097 }

```

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.

```

3098 \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).

```

3099 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
3100 { \@@_fatal:n { Preamble-forgotten } }
3101 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
3102 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
3103 { @@ _ \token_to_str:N \hline : }
3104 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
3105 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }

```

```

3106 { @@ _ \token_to_str:N \hline : }
3107 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
3108 { @@ _ \token_to_str:N \hline : }
3109 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
3110 { @@ _ \token_to_str:N \hline : }
3111 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3112 { @@ _ \token_to_str:N \hline : }
3113 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3114 { \@@_fatal:n { NiceTabularX~probably~required } }
3115 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3116 { @@ _ \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).

```

3117 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3118 {

```

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`.

```

3119 \multispan { #1 }
3120 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3121 \begingroup
3122 \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3123 \tl_gclear:N \g_@@_preamble_tl
3124 \@@_make_m_preamble:n #2 \q_stop

```

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

```

3125 \def \@addamp
3126 {
3127 \legacy_if:nTF { @firstamp }
3128 { \legacy_if_set_false:n { @firstamp } }
3129 { \@preamerr 5 }
3130 }
3131 \exp_args:No \@mkpream \g_@@_preamble_tl
3132 \@addtopreamble \@empty
3133 \endgroup
3134 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3135 \int_compare:nNnT { #1 } > 1
3136 {
3137 \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3138 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3139 \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3140 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3141 {
3142 {
3143 \int_if_zero:nTF \c@jCol
3144 { \int_eval:n { \c@iRow + 1 } }
3145 { \int_use:N \c@iRow }
3146 }
3147 { \int_eval:n { \c@jCol + 1 } }
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 } }

```

The last argument is for the name of the block.

```

3154     { }
3155 }
3156 }

```

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

```

3157 \RenewDocumentCommand { \cellcolor } { 0 { } m }
3158 {
3159     \tl_gput_right:Ne \g_@@_pre_code_before_tl
3160     {
3161         \@@_rectanglecolor [ ##1 ]
3162         { \exp_not:n { ##2 } }
3163         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3164         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3165     }
3166     \ignorespaces
3167 }

```

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

```

3168 \def \@sharp { #3 }
3169 \@arstrut
3170 \@preamble
3171 \null

```

We add some lines.

```

3172 \int_gadd:Nn \c@jCol { #1 - 1 }
3173 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
3174 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3175 \ignorespaces
3176 }

```

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`.

```

3177 \
3178 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3179 {
3180     \str_case:nnF { #1 }
3181     {
3182         c { \@@_make_m_preamble_i:n #1 }
3183         l { \@@_make_m_preamble_i:n #1 }
3184         X { \@@_make_m_preamble_i:n l }
3185         r { \@@_make_m_preamble_i:n #1 }
3186         > { \@@_make_m_preamble_ii:nn #1 }
3187         ! { \@@_make_m_preamble_ii:nn #1 }
3188         @ { \@@_make_m_preamble_ii:nn #1 }
3189         | { \@@_make_m_preamble_iii:n #1 }
3190         p { \@@_make_m_preamble_iv:nnn t #1 }
3191         m { \@@_make_m_preamble_iv:nnn c #1 }
3192         b { \@@_make_m_preamble_iv:nnn b #1 }
3193         w { \@@_make_m_preamble_v:nnnn { } #1 }
3194         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3195         \q_stop { }
3196     }
3197 {
3198     \cs_if_exist:cTF { NC @ find @ #1 }

```

```

3199     {
3200       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3201       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3202     }
3203     {
3204       \str_if_eq:nnTF { #1 } { S }
3205       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3206       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3207     }
3208   }
3209 }

```

For c, l and r

```

3210 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3211 {
3212   \tl_gput_right:Nn \g_@@_preamble_tl
3213   {
3214     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3215     #1
3216     < \@@_cell_end:
3217   }

```

We test for the presence of a <.

```

3218   \@@_make_m_preamble_x:n
3219 }

```

For >, ! and @

```

3220 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3221 {
3222   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3223   \@@_make_m_preamble:n
3224 }

```

For l

```

3225 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3226 {
3227   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3228   \@@_make_m_preamble:n
3229 }

```

For p, m and b

```

3230 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3231 {
3232   \tl_gput_right:Nn \g_@@_preamble_tl
3233   {
3234     > {
3235       \@@_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`.

```

3236   \setlength { \l_tmpa_dim } { #3 }
3237   \begin { minipage } [ #1 ] { \l_tmpa_dim }
3238   \mode_leave_vertical:
3239   \arraybackslash
3240   \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3241   }
3242   c
3243   < {
3244     \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3245     \end { minipage }
3246     \@@_cell_end:
3247   }
3248 }

```

We test for the presence of a <.

```
3249 \@@_make_m_preamble_x:n
3250 }
```

For w and W

```
3251 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3252 {
3253   \tl_gput_right:Nn \g_@@_preamble_tl
3254   {
3255     > {
3256       \dim_set:Nn \l_@@_col_width_dim { #4 }
3257       \hbox_set:Nw \l_@@_cell_box
3258       \@@_cell_begin:
3259       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3260     }
3261     c
3262     < {
3263       \@@_cell_end:
3264       \hbox_set_end:
3265       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3266       #1
3267       \@@_adjust_size_box:
3268       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3269     }
3270   }
```

We test for the presence of a <.

```
3271 \@@_make_m_preamble_x:n
3272 }
```

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

```
3273 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3274 {
3275   \str_if_eq:nnTF { #1 } { < }
3276   \@@_make_m_preamble_ix:n
3277   { \@@_make_m_preamble:n { #1 } }
3278 }
3279 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3280 {
3281   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3282   \@@_make_m_preamble_x:n
3283 }
```

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).

```
3284 \cs_new_protected:Npn \@@_put_box_in_flow:
3285 {
3286   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3287   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3288   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3289   { \box_use_drop:N \l_tmpa_box }
3290   \@@_put_box_in_flow_i:
3291 }
```

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

```
3292 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3293 {
3294   \pgfpicture
3295   \@@_qpoint:n { row - 1 }
```

```

3296 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3297 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3298 \dim_gadd:Nn \g_tmpa_dim \pgf@y
3299 \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).

```

3300 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3301 {
3302   \int_set:Nn \l_tmpa_int
3303   { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3304   \bool_lazy_or:nnT
3305   { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3306   { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3307   {
3308     \@@_error:n { bad-value-for-baseline-line }
3309     \int_set:Nn \l_tmpa_int 1
3310   }
3311   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3312 }
3313 {
3314   \str_if_eq:eeTF t \l_@@_baseline_tl
3315   { \int_set:Nn \l_tmpa_int 1 }
3316   {
3317     \str_if_eq:eeTF b \l_@@_baseline_tl
3318     { \int_set_eq:NN \l_tmpa_int \c@iRow }
3319     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3320   }
3321   \bool_lazy_or:nnT
3322   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3323   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3324   {
3325     \@@_error:n { bad-value-for-baseline }
3326     \int_set:Nn \l_tmpa_int 1
3327   }
3328   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3329 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3330 }
3331 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3332 \endpgfpicture
3333 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3334 }

```

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).

```

3335 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3336 {

```

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.

```

3337 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3338 {
3339   \int_compare:nNnT \c@jCol > 1
3340   {
3341     \box_set_wd:Nn \l_@@_the_array_box
3342     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3343   }
3344 }

```

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

```

3345 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3346 \bool_if:NT \l_@@_caption_above_bool
3347 {
3348   \tl_if_empty:NF \l_@@_caption_tl
3349   {
3350     \bool_set_false:N \g_@@_caption_finished_bool
3351     \int_gzero:N \c@tabularnote
3352     \@@_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).

```

3353     \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3354     {
3355       \tl_gput_right:Ne \g_@@_aux_tl
3356       {
3357         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3358         { \int_use:N \g_@@_notes_caption_int }
3359       }
3360       \int_gzero:N \g_@@_notes_caption_int
3361     }
3362   }
3363 }

```

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

```

3364 \hbox
3365 {
3366   \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.

```

3367 \@@_create_extra_nodes:
3368 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3369 }

```

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).

```

3370 \bool_lazy_any:nT
3371 {
3372   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3373   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3374   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3375 }
3376 {
3377   \bool_if:NTF \l_@@_notes_no_print_bool
3378   { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3379   \@@_tabular_notes:
3380 }
3381 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3382 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3383 \end { minipage }
3384 }

```

```

3385 \cs_new_protected:Npn \@@_insert_caption:
3386 {
3387   \tl_if_empty:NF \l_@@_caption_tl
3388   {

```

```

3389     \cs_if_exist:NTF \@cptype
3390     { \@@_insert_caption_i: }
3391     { \@@_error:n { caption~outside~float } }
3392   }
3393 }

```

```

3394 \cs_new_protected:Npn \@@_insert_caption_i:
3395 {
3396   \group_begin:

```

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

```

3397   \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.

```

3398   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3399   \tl_if_empty:NTF \l_@@_short_caption_tl
3400     \caption
3401     { \caption [ \l_@@_short_caption_tl ] }
3402     { \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.

```

3403   \bool_if:NF \g_@@_caption_finished_bool
3404   {
3405     \bool_gset_true:N \g_@@_caption_finished_bool
3406     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3407     \int_gzero:N \c@tabularnote
3408   }
3409   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3410   \group_end:
3411 }
3412 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3413 {
3414   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3415   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3416 }
3417 \cs_set_protected:Npn \@@_tabular_notes_error:
3418 { \@@_error:n { Misuse~of~NiceTabularNotes } }
3419 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3420 \cs_set_protected:Npn \@@_tabular_notes:
3421 {
3422   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3423   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3424   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3425   \skip_vertical:N 0.65ex

```

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

```

3426   \group_begin:
3427   \l_@@_notes_code_before_tl
3428   \tl_if_empty:NF \g_@@_tabularnote_tl
3429   {
3430     \g_@@_tabularnote_tl \par
3431     \tl_gclear:N \g_@@_tabularnote_tl
3432   }

```

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.

```

3433 \int_compare:nNnT \c@tabularnote > \c_zero_int
3434 {
3435   \bool_if:NTF \l_@@_notes_para_bool
3436   {
3437     \begin { tabularnotes* }
3438     \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3439     \strut
3440     \end { tabularnotes* }

```

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

```

3441 \par
3442 }
3443 {
3444   \tabularnotes
3445   \seq_map_inline:Nn \g_@@_notes_seq { \@@_one_tabularnote:nn ##1 }
3446   \strut
3447   \endtabularnotes
3448 }
3449 }
3450 \unskip
3451 \group_end:
3452 \bool_if:NT \l_@@_notes_bottomrule_bool
3453 {
3454   \IfPackageLoadedTF { booktabs }
3455   {

```

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

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

```

3456 \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
3457 { \CT@arc@ \hrule height \heavyrulewidth }
3458 }
3459 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3460 }
3461 \l_@@_notes_code_after_tl
3462 \seq_gclear:N \g_@@_notes_seq
3463 \seq_gclear:N \g_@@_notes_in_caption_seq
3464 \int_gzero:N \c@tabularnote
3465 }

```

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 currying.

```

3466 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3467 {
3468   \tl_if_novalue:nTF { #1 }
3469   { \item }
3470   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3471 }

```

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`.

```

3472 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3473 {
3474   \pgfpicture
3475   \@@_qpoint:n { row - 1 }
3476   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3477   \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3478   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

```

3479 \endpgfpicture
3480 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3481 \int_if_zero:nT \l_@@_first_row_int
3482 {
3483     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3484     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3485 }
3486 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3487 }

```

Now, the general case.

```

3488 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3489 {

```

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

```

3490 \str_if_eq:eeT \l_@@_baseline_tl { t }
3491 { \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`.

```

3492 \pgfpicture
3493 \@@_qpoint:n { row - 1 }
3494 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3495 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3496 {
3497     \int_set:Nn \l_tmpa_int
3498     {
3499         \str_range:Nnn
3500         \l_@@_baseline_tl
3501         { 6 }
3502         { \tl_count:o \l_@@_baseline_tl }
3503     }
3504     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3505 }
3506 {
3507     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3508     \bool_lazy_or:nnT
3509     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3510     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3511     {
3512         \@@_error:n { bad-value-for-baseline }
3513         \int_set:Nn \l_tmpa_int 1
3514     }
3515     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3516 }
3517 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3518 \endpgfpicture
3519 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3520 \int_if_zero:nT \l_@@_first_row_int
3521 {
3522     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3523     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3524 }
3525 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3526 }

```

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.

```

3527 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3528 {

```

We will compute the real width of both delimiters used.

```

3529 \dim_zero_new:N \l_@@_real_left_delim_dim
3530 \dim_zero_new:N \l_@@_real_right_delim_dim
3531 \hbox_set:Nn \l_tmpb_box
3532 {
3533   \m@th
3534   $ % $
3535   \left #1
3536   \vcenter
3537   {
3538     \vbox_to_ht:nn
3539     { \box_ht_plus_dp:N \l_tmpa_box }
3540     { }
3541   }
3542   \right .
3543   $ % $
3544 }
3545 \dim_set:Nn \l_@@_real_left_delim_dim
3546 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3547 \hbox_set:Nn \l_tmpb_box
3548 {
3549   \m@th
3550   $ % $
3551   \left .
3552   \vbox_to_ht:nn
3553   { \box_ht_plus_dp:N \l_tmpa_box }
3554   { }
3555   \right #2
3556   $ % $
3557 }
3558 \dim_set:Nn \l_@@_real_right_delim_dim
3559 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3560 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3561 \@@_put_box_in_flow:
3562 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3563 }

```

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).

```

3564 \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).

```

3565 {
3566   \peek_remove_spaces:n
3567   {
3568     \peek_meaning:NTF \end
3569     \@@_analyze_end:Nn
3570     {
3571       \@@_transform_preamble:
3572       \@@_array:o \l_@@_array_preamble_tl
3573     }
3574   }
3575 }
3576 {
3577   \@@_create_col_nodes:

```

```

3578     \endarray
3579 }

```

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

```

3580 \NewDocumentEnvironment { @@-light-syntax } { b }
3581 {

```

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`.

```

3582     \tl_if_empty:nT { #1 }
3583     { \@@_fatal:n { empty-environment } }
3584     \tl_if_in:nnT { #1 } { & }
3585     { \@@_fatal:n { ampersand-in-light-syntax } }
3586     \tl_if_in:nnT { #1 } { \ }
3587     { \@@_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`.

```

3588     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3589 }

```

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.

```

3590 {
3591     \@@_create_col_nodes:
3592     \endarray
3593 }
3594 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3595 {
3596     \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).

```

3597     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3598     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3599     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3600     { \seq_set_split:Nee
3601       \seq_set_split:Non
3602       \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3603     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3604     \tl_if_empty:NF \l_tmpa_tl
3605     { \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.

```

3606     \int_compare:nNnT \l_@@_last_row_int = { -1 }
3607     { \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`.

```

3608     \tl_build_begin:N \l_@@_new_body_tl
3609     \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```
3610 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3611 \@@_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).

```
3612 \seq_map_inline:Nn \l_@@_rows_seq
3613 {
3614   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3615   \@@_line_with_light_syntax:n { ##1 }
3616 }
3617 \tl_build_end:N \l_@@_new_body_tl
3618 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3619 {
3620   \int_set:Nn \l_@@_last_col_int
3621     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3622 }
```

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.

```
3623 \@@_transform_preamble:

3624 \@@_array:o \l_@@_array_preamble_tl \l_@@_new_body_tl
3625 }
3626 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3627 {
3628   \seq_clear_new:N \l_@@_cells_seq
3629   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3630   \int_set:Nn \l_@@_nb_cols_int
3631     { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3632   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3633   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3634   \seq_map_inline:Nn \l_@@_cells_seq
3635     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3636 }
3637 \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`.

```
3638 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3639 {
3640   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3641     { \@@_fatal:n { empty-environment } } }
```

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

```
3642 \end { #2 }
3643 }
```

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

```
3644 \socket_new:nn { nicematrix / horizontal-space } { 0 }
3645 \socket_new_plug:nnn { nicematrix / horizontal-space } { standard }
3646   { \skip_horizontal:N \g_tmpa_skip }
3647 \socket_assign_plug:nn { nicematrix / horizontal-space } { standard }
3648 \socket_new_plug:nnn { nicematrix / horizontal-space } { with-false-col }
3649 {
3650   \clist_if_in:NnTF \g_@@_false_cols_clist { \int_use:N \g_tmpa_int }
3651   {
3652     % modified 2026/09/03
3653     \skip_set:Nn \l_tmpa_skip { 0 pt+plus 1 fill }
3654     % \skip_set:Nn \l_tmpa_skip { 0 pt }
```

```

3655 \skip_add:Nn \l_tmpa_skip \l_@@_width_of_false_dim
3656 \skip_add:Nn \l_tmpa_skip \arrayrulewidth
3657 \skip_horizontal:N \l_tmpa_skip
3658 }
3659 { \skip_horizontal:N \g_tmpa_skip }
3660 }

```

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`).

```

3661 \cs_new:Npn \@@_create_col_nodes:
3662 {
3663   \crr
3664   \int_if_zero:nT \l_@@_first_col_int
3665   {
3666     \omit
3667     \hbox_overlap_left:n
3668     {
3669       \bool_if:NT \l_@@_code_before_bool
3670       { \pgfsys@markposition { \@@_env: - col - 0 } }
3671       \pgfpicture
3672       \pgfrememberpicturerepositiononpagetrue
3673       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3674       \str_if_empty:NF \l_@@_name_str
3675       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3676       \endpgfpicture
3677       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3678     }
3679     &
3680   }
3681   \omit

```

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

```

3682 \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`).

```

3683 \int_if_zero:nTF \l_@@_first_col_int
3684 {
3685   \@@_mark_position:n { 1 }
3686   \pgfpicture
3687   \pgfrememberpicturerepositiononpagetrue
3688   \pgfcoordinate { \@@_env: - col - 1 }
3689   { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3690   \str_if_empty:NF \l_@@_name_str
3691   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3692   \endpgfpicture
3693 }
3694 {
3695   \bool_if:NT \l_@@_code_before_bool
3696   {
3697     \hbox
3698     {
3699       \skip_horizontal:n { 0.5 \arrayrulewidth }
3700       \pgfsys@markposition { \@@_env: - col - 1 }
3701       \skip_horizontal:n { -0.5 \arrayrulewidth }
3702     }
3703   }
3704   \pgfpicture
3705   \pgfrememberpicturerepositiononpagetrue
3706   \pgfcoordinate { \@@_env: - col - 1 }
3707   { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3708   \@@_node_alias:n { 1 }

```

```

3709     \endpgfpicture
3710 }

```

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.

```

3711 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3712 \bool_if:NTF \l_@@_row_columns_width_bool
3713 { \skip_gadd:Nn \g_tmpa_skip { \box_ht_plus_dp:N \@arstrutbox } }
3714 {
3715   \bool_lazy_or:nnT
3716   { \l_@@_auto_columns_width_bool }
3717   { \dim_compare_p:nNn \l_@@_columns_width_dim > \c_zero_dim }
3718   {
3719     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3720     \bool_lazy_and:nnTF
3721     \l_@@_auto_columns_width_bool
3722     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3723     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3724     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3725   }
3726 }
3727 \int_gset:Nn \g_tmpa_int 1
3728 \socket_use:n { nicematrix / horizontal-space }
3729 \hbox
3730 {
3731   \@@_mark_position:n { 2 }
3732   \pgfpicture
3733   \pgfrememberpicturepositiononpagetrue
3734   \pgfcoordinate { \@@_env: - col - 2 }
3735   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3736   \@@_node_alias:n { 2 }
3737   \endpgfpicture
3738 }

```

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.

```

3739 \bool_if:NTF \g_@@_last_col_found_bool
3740 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3741 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3742 {
3743   &
3744   \omit
3745   \int_gincr:N \g_tmpa_int

```

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

```

3746   \socket_use:n { nicematrix / horizontal-space }
3747   \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3748   \pgfpicture
3749   \pgfrememberpicturepositiononpagetrue
3750   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3751   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3752   \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3753   \endpgfpicture
3754 }

```

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).

```

3755   \bool_lazy_or:nnF
3756   { \int_compare_p:nNn \g_@@_col_total_int = 1 }

```

```

3757 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3758 {
3759   &
3760   \omit
3761   \skip_horizontal:N \g_tmpa_skip
3762   \int_gincr:N \g_tmpa_int
3763   \bool_lazy_any:nF
3764   {
3765     \g_@@_delims_bool
3766     \l_@@_tabular_bool
3767     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3768     \l_@@_exterior_arraycolsep_bool
3769     \l_@@_bar_at_end_of_pream_bool
3770   }
3771   { \skip_horizontal:n { - \col@sep } }
3772   \bool_if:NT \l_@@_code_before_bool
3773   {
3774     \hbox
3775     {
3776       \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.

```

3777       \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3778       { \skip_horizontal:n { - \arraycolsep } }
3779       \pgfsys@markposition
3780       { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3781       \skip_horizontal:n { 0.5 \arrayrulewidth }
3782       \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3783       { \skip_horizontal:N \arraycolsep }
3784     }
3785   }
3786   \pgfpicture
3787   \pgfrememberpicturepositiononpagetrue
3788   \pgfcoordinate { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3789   {
3790     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3791     {
3792       \pgfpoint
3793       { - 0.5 \arrayrulewidth - \arraycolsep }
3794       \c_zero_dim
3795     }
3796     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3797   }
3798   @@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3799   \endpgfpicture
3800 }

```

```

3801 \bool_if:NT \g_@@_last_col_found_bool
3802 {
3803   \hbox_overlap_right:n
3804   {
3805     \skip_horizontal:N \g_@@_width_last_col_dim
3806     \skip_horizontal:N \col@sep
3807     \bool_if:NT \l_@@_code_before_bool
3808     {
3809       \pgfsys@markposition
3810       { @@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3811     }
3812     \pgfpicture
3813     \pgfrememberpicturepositiononpagetrue
3814     \pgfcoordinate

```

```

3815         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3816         \pgfpintorigin
3817         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3818         \endpgfpicture
3819     }
3820 }
3821 }

3822 \cs_new_protected:Npn \@@_mark_position:n #1
3823 {
3824     \bool_if:NT \l_@@_code_before_bool
3825     {
3826         \hbox
3827         {
3828             \skip_horizontal:n { -0.5 \arrayrulewidth }
3829             \pgfsys@markposition { \@@_env: - col - #1 }
3830             \skip_horizontal:n { 0.5 \arrayrulewidth }
3831         }
3832     }
3833 }

3834 \cs_new_protected:Npn \@@_node_alias:n #1
3835 {
3836     \str_if_empty:NF \l_@@_name_str
3837     { \pgfnodelalias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3838 }

```

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

```

3839 \tl_const:Nn \c_@@_preamble_first_col_tl
3840 {
3841     >
3842     {

```

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

```

3843         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3844         \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3845         \cs_set_eq:NN \FalseRow \@@_FalseRow_in_cell:
3846         \bool_gset_true:N \g_@@_after_col_zero_bool
3847         \@@_begin_of_row:
3848         \hbox_set:Nw \l_@@_cell_box
3849         \@@_math_toggle:
3850         \@@_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”.

```

3851         \int_compare:nNnT \c@iRow > \c_zero_int
3852         {
3853             \bool_lazy_or:nnT
3854             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3855             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3856             {
3857                 \l_@@_code_for_first_col_tl
3858                 \xglobal \colorlet { nicematrix-first-col } { . }
3859             }
3860         }
3861     }

```

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`.

```

3862     l
3863     <

```

```

3864 {
3865   \@@_math_toggle:
3866   \hbox_set_end:
3867   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3868   \@@_adjust_size_box:
3869   \@@_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.

```

3870   \dim_gset:Nn \g_@@_width_first_col_dim
3871   { \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.

```

3872   \hbox_overlap_left:n
3873   {
3874     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3875     \@@_node_cell:
3876     { \box_use_drop:N \l_@@_cell_box }
3877     \skip_horizontal:N \l_@@_left_delim_dim
3878     \skip_horizontal:N \l_@@_left_margin_dim
3879     \skip_horizontal:N \l_@@_extra_left_margin_dim
3880   }
3881   \bool_gset_false:N \g_@@_empty_cell_bool
3882   \skip_horizontal:n { -2 \col@sep }
3883 }
3884 }

```

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

```

3885 \tl_const:Nn \c_@@_preamble_last_col_tl
3886 {
3887   >
3888   {
3889     \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).

```

3890     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3891     \cs_set_eq:NN \Hline \@@_Hline_in_cell:
3892     \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.

```

3893     \bool_gset_true:N \g_@@_last_col_found_bool
3894     \int_gincr:N \c@jCol
3895     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3896     \hbox_set:Nw \l_@@_cell_box
3897     \@@_math_toggle:
3898     \@@_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”.

```

3899     \int_compare:nNnT \c@iRow > \c_zero_int
3900     {
3901       \bool_lazy_or:nnT
3902       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3903       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3904       {
3905         \l_@@_code_for_last_col_tl
3906         \xglobal \colorlet { nicematrix-last-col } { . }
3907       }
3908     }
3909   }
3910   1
3911   <
3912   {
3913     \@@_math_toggle:
3914     \hbox_set_end:

```

```

3915 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3916 \@@_adjust_size_box:
3917 \@@_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.

```

3918 \dim_gset:Nn \g_@@_width_last_col_dim
3919 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3920 \skip_horizontal:n { -2 \col@sep }

```

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

```

3921 \hbox_overlap_right:n
3922 {
3923   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3924   {
3925     \skip_horizontal:N \l_@@_right_delim_dim
3926     \skip_horizontal:N \l_@@_right_margin_dim
3927     \skip_horizontal:N \l_@@_extra_right_margin_dim
3928     \@@_node_cell:
3929   }
3930 }
3931 \bool_gset_false:N \g_@@_empty_cell_bool
3932 }
3933 }

```

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

```

3934 \NewDocumentEnvironment { NiceArray } { }
3935 {
3936   \bool_gset_false:N \g_@@_delims_bool
3937   \str_if_empty:NT \g_@@_name_env_str
3938   { \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).

```

3939   \NiceArrayWithDelims . .
3940 }
3941 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3942 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3943 {
3944   \NewDocumentEnvironment { #1 NiceArray } { }
3945   {
3946     \bool_gset_true:N \g_@@_delims_bool
3947     \str_if_empty:NT \g_@@_name_env_str
3948     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3949     \@@_test_if_math_mode:
3950     \NiceArrayWithDelims #2 #3
3951   }
3952   { \endNiceArrayWithDelims }
3953 }
3954 \@@_def_env:NNN p ( )
3955 \@@_def_env:NNN b [ ]
3956 \@@_def_env:NNN B \{ \}
3957 \@@_def_env:NNN v \vert \vert
3958 \@@_def_env:NNN V \Vert \Vert

```

13 The environment {NiceMatrix} and its variants

13.1 Definition of {pNiceMatrix}

```

3959 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3960 {
3961   \bool_set_false:N \l_@@_preamble_bool
3962   \tl_clear:N \l_tmpa_tl
3963   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3964     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3965   \tl_put_right:Nn \l_tmpa_tl
3966     {
3967       *
3968       {
3969         \int_case:nnF \l_@@_last_col_int
3970         {
3971           { -2 } { \c@MaxMatrixCols }
3972           { -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).

```

3973       }
3974       { \int_eval:n { \l_@@_last_col_int - 1 } }
3975     }
3976     { #2 }
3977   }
3978   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3979   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3980 }
3981 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3982 \clist_map_inline:nn { p , b , B , v , V }
3983 {
3984   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3985   {
3986     \bool_gset_true:N \g_@@_delims_bool
3987     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3988     \int_if_zero:nT \l_@@_last_col_int
3989     {
3990       \bool_set_true:N \l_@@_last_col_without_value_bool
3991       \int_set:Nn \l_@@_last_col_int { -1 }
3992     }
3993     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3994     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3995   }
3996   { \use:c { end #1 NiceArray } }
3997 }

```

We define also an environment {NiceMatrix}

```

3998 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3999 {
4000   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
4001   \int_if_zero:nT \l_@@_last_col_int
4002   {
4003     \bool_set_true:N \l_@@_last_col_without_value_bool
4004     \int_set:Nn \l_@@_last_col_int { -1 }
4005   }
4006   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
4007   \bool_lazy_or:nnT
4008     \l_@@_except_borders_bool
4009     { \clist_if_empty_p:N \l_@@_vlines_clist }
4010     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
4011   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
4012 }

```

```
4013 { \endNiceArray }
```

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

```
4014 \cs_new_protected:Npn \@@_NotEmpty:
4015 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

13.2 The key renew-matrix

```
4016 \cs_set_protected:Npn \@@_renew_matrix:
4017 {
4018   \tl_map_inline:nn { pvVbB }
4019   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
4020 }
```

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

```
4021 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
4022 {
```

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.

```
4023   \dim_compare:nNtT\l_@@_width_dim = \c_zero_dim
4024   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
4025   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
4026   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
4027   \tl_if_empty:NF \l_@@_short_caption_tl
4028   {
4029     \tl_if_empty:NT \l_@@_caption_tl
4030     {
4031       \@@_error_or_warning:n { short-caption~without~caption }
4032       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
4033     }
4034   }
4035   \tl_if_empty:NF \l_@@_label_tl
4036   {
4037     \tl_if_empty:NT \l_@@_caption_tl
4038     { \@@_error_or_warning:n { label~without~caption } }
4039   }
4040   \NewDocumentEnvironment { TabularNote } { b }
4041   {
4042     \bool_if:NTF \l_@@_in_code_after_bool
4043     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
4044     {
4045       \tl_if_empty:NF \g_@@_tabularnote_tl
4046       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
4047       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
4048     }
4049   }
4050   { }
4051   \@@_settings_for_tabular:
4052   \NiceArray { #2 }
4053 }
4054 { \endNiceArray }
4055 \cs_new_protected:Npn \@@_settings_for_tabular:
4056 {
4057   \bool_set_true:N \l_@@_tabular_bool
4058   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
4059   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
4060   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
4061 }
```

```

4062 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
4063 {
4064   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
4065   \dim_set:Nn \l_@@_width_dim { #1 }
4066   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4067   \@@_settings_for_tabular:
4068   \NiceArray { #3 }
4069 }
4070 {
4071   \endNiceArray
4072   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
4073   { \@@_error_or_warning:n { NiceTabularX~without~X } }
4074 }

4075 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
4076 {
4077   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
4078   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
4079   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
4080   \@@_settings_for_tabular:
4081   \NiceArray { #3 }
4082 }
4083 { \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`).

```

4084 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
4085 {
4086   \bool_lazy_all:nT
4087   {
4088     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
4089     { \l_@@_hvlines_bool }
4090     { ! \g_@@_delims_bool }
4091     { ! \l_@@_except_borders_bool }
4092   }
4093   {
4094     \bool_set_true:N \l_@@_except_borders_bool
4095     \clist_if_empty:NF \l_@@_corners_clist
4096     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
4097     \tl_gput_right:Nn \g_@@_rules_tl
4098     {
4099       \@@_stroke_block:nnnnn
4100       {
4101         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
4102         draw = \l_@@_rules_color_tl
4103       }
4104       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
4105     }
4106   }
4107 }

4108 \cs_new_protected:Npn \@@_after_array:
4109 {
4110   \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.

```
4111 \bool_if:NT \g_@@_last_col_found_bool
4112 { \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`.

```
4113 \bool_if:NT \l_@@_last_col_without_value_bool
4114 { \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.

```
4115 \bool_if:NT \l_@@_last_row_without_value_bool
4116 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
4117 \tl_gput_right:Ne \g_@@_aux_tl
4118 {
4119   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
4120   {
4121     \int_use:N \l_@@_first_row_int ,
4122     \int_use:N \c@iRow ,
4123     \int_use:N \g_@@_row_total_int ,
4124     \int_use:N \l_@@_first_col_int ,
4125     \int_use:N \c@jCol ,
4126     \int_use:N \g_@@_col_total_int
4127   }
4128 }
4129 \clist_if_empty:NF \g_@@_false_cols_clist
4130 {
4131   \tl_gput_right:Ne \g_@@_aux_tl
4132   {
4133     \clist_set:Nn \exp_not:N \g_@@_false_cols_clist
4134     { \g_@@_false_cols_clist }
4135   }
4136 }
4137 \clist_if_empty:NF \g_@@_false_rows_clist
4138 {
4139   \tl_gput_right:Ne \g_@@_aux_tl
4140   {
4141     \clist_set:Nn \exp_not:N \g_@@_false_rows_clist
4142     { \g_@@_false_rows_clist }
4143   }
4144 }
4145 \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`).

```
4146 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
4147 {
4148   \tl_gput_right:Ne \g_@@_aux_tl
4149   {
4150     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4151     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4152   }
4153 }
4154 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4155 {
4156   \tl_gput_right:Ne \g_@@_aux_tl
```

```

4157     {
4158         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4159         { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4160         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4161         { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4162     }
4163 }

```

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

```

4164 \@@_create_diag_nodes:

```

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

```

4165 \pgfpicture
4166 \@@_create_aliases_last:
4167 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4168 \endpgfpicture

4169 \bool_lazy_and:nnF
4170 { \clist_if_empty_p:N \g_@@_false_cols_clist }
4171 { \clist_if_empty_p:N \g_@@_false_rows_clist }
4172 {
4173     \@@_mark_blocks:
4174     \@@_treat_false_cols:
4175     \@@_treat_false_rows:
4176 }

```

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.

```

4177 \bool_if:NT \l_@@_parallelize_diags_bool
4178 {
4179     \int_gzero:N \g_@@_ddots_int
4180     \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.

```

4181     \dim_gzero:N \g_@@_delta_x_one_dim
4182     \dim_gzero:N \g_@@_delta_y_one_dim
4183     \dim_gzero:N \g_@@_delta_x_two_dim
4184     \dim_gzero:N \g_@@_delta_y_two_dim
4185 }

4186 \bool_set_false:N \l_@@_initial_open_bool
4187 \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

4188 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4189 \@@_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.

```

4190 \clist_if_empty:NF \l_@@_corners_clist

```

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

```

4191     {
4192       \bool_if:NTF \l_@@_no_cell_nodes_bool
4193       { \@@_error:n { corners~with~no~cell~nodes } }
4194       \@@_compute_corners:
4195     }

```

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`.

```

4196     \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4197     \g_@@_pos_of_blocks_seq
4198     \g_@@_future_pos_of_blocks_seq
4199     \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-*}`).

```

4200     \@@_adjust_pos_of_blocks_seq:
4201
4202     \@@_deal_with_rounded_corners:
4203     \legacy_if:nF { measuring@ } \@@_draw_rules:
4204     \tl_gclear:N \g_@@_rules_tl

```

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

```

4204     \IfPackageLoadedT { tikz }
4205     {
4206       \tikzset
4207       {
4208         every-picture / .style =
4209         {
4210           overlay ,
4211           remember~picture ,
4212           name~prefix = \@@_env: -
4213         }
4214       }
4215     }
4216     \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4217     \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4218     \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4219     \cs_set_eq:NN \OverBrace \@@_OverBrace
4220     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4221     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4222     \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.

```

4223     \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4224     \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.

```

4225     \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).

```

4226     \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`).

```

4227     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
4228     { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }

```

```

4229 \bool_set_true:N \l_@@_in_code_after_bool
4230 \@@_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::`.

```

4231 \bool_set_true:N \l_@@_in_code_after_bool
4232 \if_mode_math:
4233 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4234 \scan_stop:
4235 \else:
4236 $ % $
4237 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4238 \scan_stop:
4239 $ % $
4240 \fi:
4241 \tl_gclear:N \g_nicematrix_code_after_tl
4242 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4243 \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.

```

4244 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4245 \tl_if_empty:NF \g_@@_pre_code_before_tl
4246 {
4247 \tl_gput_right:Ne \g_@@_aux_tl
4248 {
4249 \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4250 { \exp_not:o \g_@@_pre_code_before_tl }
4251 }
4252 \tl_gclear:N \g_@@_pre_code_before_tl
4253 }
4254 \tl_if_empty:NF \g_nicematrix_code_before_tl
4255 {
4256 \tl_gput_right:Ne \g_@@_aux_tl
4257 {
4258 \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4259 { \exp_not:o \g_nicematrix_code_before_tl }
4260 }
4261 \tl_gclear:N \g_nicematrix_code_before_tl
4262 }

4263 \str_gclear:N \g_@@_name_env_str
4264 \@@_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`.

```

4265 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4266 }

4267 \cs_new_protected:Npn \@@_test_false_columns:
4268 {
4269 \clist_map_inline:Nn \g_@@_false_cols_clist
4270 {
4271 \int_step_inline:nnn { \l_@@_first_row_int } { \arabic { iRow } }
4272 {

```

¹⁴e.g. `\color[rgb]{0.5,0.5,0}`

```

4273         \cs_if_exist:cT
4274         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
4275         {
4276             \@@_error:nnn { Not~empty~cell~in~false~column }
4277             { #####1 }
4278             { ##1 }
4279         }
4280     }
4281 }
4282 }

4283 \cs_new_protected:Npn \@@_treat_false_rows:
4284 { \clist_map_function:NN \g_@@_false_rows_clist \@@_treat_one_false_row:n }

4285 \cs_new_protected:Npn \@@_treat_one_false_row:n #1
4286 {
4287     \int_zero:N \l_tmpa_int
4288     \int_step_inline:nn \c@jCol
4289     {
4290         \int_if_zero:nTF \l_tmpa_int
4291         {
4292             \@@_if_in_block:nnF { #1 } { ##1 }
4293             { \int_set:Nn \l_tmpa_int { ##1 } }
4294         }
4295         {
4296             \@@_if_in_block:nnT { #1 } { ##1 }
4297             {
4298                 \@@_false_h_segment:nee
4299                 { #1 }
4300                 { \int_use:N \l_tmpa_int }
4301                 { \int_eval:n { ##1 - 1 } }
4302                 \int_zero:N \l_tmpa_int
4303             }
4304         }
4305     }
4306     \int_compare:nNnT \l_tmpa_int > 0
4307     {
4308         \clist_if_in:NnTF \g_@@_false_cols_clist { \int_use:N \c@jCol }
4309         { \int_set_eq:NN \l_tmpb_int \c@jCol }
4310         { \int_set:Nn \l_tmpb_int { \c@jCol + 1 } }
4311         \@@_false_h_segment:nee { #1 }
4312         { \int_use:N \l_tmpa_int }
4313         { \int_use:N \l_tmpb_int }
4314     }
4315 }

4316 \cs_new_protected:Npn \@@_false_h_segment:nnn #1 #2 #3
4317 {

```

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

```

4318     \int_compare:nNnTF { #2 } = 1
4319     {
4320         \clist_if_in:NnTF \g_@@_false_cols_clist { 1 }
4321         { \int_set:Nn \l_tmpa_int 1 }
4322         { \int_zero:N \l_tmpa_int }
4323     }
4324     { \int_set:Nn \l_tmpa_int { #2 } }
4325     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4326     {
4327         { #1 }

```

```

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

4371 \cs_new_protected:Npn \@@_false_v_segment:nnn #1 #2 #3
4372 {
4373   \int_compare:nNnTF { #2 } = 1
4374   {
4375     \clist_if_in:NnTF \g_@@_false_rows_clist { 1 }
4376     { \int_set:Nn \l_tmpa_int 1 }
4377     { \int_zero:N \l_tmpa_int }
4378   }
4379   { \int_set:Nn \l_tmpa_int { #2 } }
4380   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
4381   {
4382     { \int_use:N \l_tmpa_int }
4383     { #1 }
4384     { #3 }
4385     { #1 }
4386     { }
4387   }
4388   \tl_gput_right:Ne \g_@@_pre_code_before_tl

```

```

4389     { \l_@@_rectanglecolor { \l_@@_color_of_false_tl } { #2 - #1 } { #3 - #1 } }
4390   }
4391   \cs_generate_variant:Nn \l_@@_false_v_segment:nnn { n e e }

```

```

4392 \cs_new_protected:Npn \l_@@_tuning_key_small_for_dots:
4393 {
4394   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4395   \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.

```

4396   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4397     { 0.6 \l_@@_xdots_shorten_start_dim }
4398   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4399     { 0.6 \l_@@_xdots_shorten_end_dim }
4400 }

```

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`.

```

4401 \NewDocumentCommand \l_@@_CodeAfter_keys: { 0 { } }
4402 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

4403 \cs_new_protected:Npn \l_@@_create_alias_nodes:
4404 {
4405   \int_step_inline:nn \c@iRow
4406   {
4407     \pgfnodealias
4408       { \l_@@_name_str - ##1 - last }
4409       { \l_@@_env: - ##1 - \int_use:N \c@jCol }
4410   }
4411   \int_step_inline:nn \c@jCol
4412   {
4413     \pgfnodealias
4414       { \l_@@_name_str - last - ##1 }
4415       { \l_@@_env: - \int_use:N \c@iRow - ##1 }
4416   }
4417   \pgfnodealias
4418     { \l_@@_name_str - last - last }
4419     { \l_@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4420 }

```

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).

```

4421 \cs_new_protected:Npn \l_@@_adjust_pos_of_blocks_seq:
4422 {
4423   \seq_gset_map_e:Nnn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4424     { \l_@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4425 }

```

The following command must *not* be protected.

```

4426 \cs_new:Npn \l_@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4427 {
4428   { #1 }
4429   { #2 }

```

```

4430 {
4431   \int_compare:nNnTF { #3 } > { 98 }
4432   { \int_use:N \c@iRow }
4433   { #3 }
4434 }
4435 {
4436   \int_compare:nNnTF { #4 } > { 98 }
4437   { \int_use:N \c@jCol }
4438   { #4 }
4439 }
4440 { #5 }
4441 }

```

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).

```

4442 \AtBeginDocument
4443 {
4444   \cs_new_protected:Npe \@@_draw_dotted_lines:
4445   {
4446     \c_@@_pgfortikzpicture_tl
4447     \@@_draw_dotted_lines_i:
4448     \c_@@_endpgfortikzpicture_tl
4449   }
4450 }

```

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

```

4451 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4452 {
4453   \pgfrememberpicturepositiononpagetrue
4454   \pgf@relevantforpicturesizefalse
4455   \g_@@_HVdotsfor_lines_tl
4456   \g_@@_Vdots_lines_tl
4457   \g_@@_Ddots_lines_tl
4458   \g_@@_Iddots_lines_tl
4459   \g_@@_Cdots_lines_tl
4460   \g_@@_Ldots_lines_tl
4461 }

4462 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4463 {
4464   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4465   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4466 }

```

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

```

4467 \pgfdeclareshape { @@_diag_node }
4468 {
4469   \savedanchor { \five }
4470   {
4471     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4472     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4473   }
4474   \anchor { 5 } { \five }
4475   \anchor { center } { \pgfpointorigin }
4476   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4477   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4478   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4479   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }

```

```

4480 \anchor { 4 } { \five \pgf{x = 0.8 \pgf{x \pgf{y = 0.8 \pgf{y }
4481 \anchor { 6 } { \five \pgf{x = 1.2 \pgf{x \pgf{y = 1.2 \pgf{y }
4482 \anchor { 7 } { \five \pgf{x = 1.4 \pgf{x \pgf{y = 1.4 \pgf{y }
4483 \anchor { 75 } { \five \pgf{x = 1.5 \pgf{x \pgf{y = 1.5 \pgf{y }
4484 \anchor { 8 } { \five \pgf{x = 1.6 \pgf{x \pgf{y = 1.6 \pgf{y }
4485 \anchor { 9 } { \five \pgf{x = 1.8 \pgf{x \pgf{y = 1.8 \pgf{y }
4486 }

```

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).

```

4487 \cs_new_protected:Npn \@@_create_diag_nodes:
4488 {
4489 \pgfpicture
4490 \pgfrememberpicturepositiononpagetrue
4491 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4492 {
4493 \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4494 \dim_set_eq:NN \l_tmpa_dim \pgf{x
4495 \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4496 \dim_set_eq:NN \l_tmpb_dim \pgf{y
4497 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4498 \dim_set_eq:NN \l_@@_tmpc_dim \pgf{x
4499 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4500 \dim_set_eq:NN \l_@@_tmpd_dim \pgf{y
4501 \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.

```

4502 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4503 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4504 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4505 \str_if_empty:NF \l_@@_name_str
4506 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4507 }

```

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

```

4508 \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4509 \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4510 \dim_set_eq:NN \l_tmpa_dim \pgf{y
4511 \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4512 \pgfcoordinate
4513 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf{x \l_tmpa_dim }
4514 \pgfnodealias
4515 { \@@_env: - last }
4516 { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4517 \str_if_empty:NF \l_@@_name_str
4518 {
4519 \pgfnodealias
4520 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4521 { \@@_env: - \int_use:N \l_tmpa_int }
4522 \pgfnodealias
4523 { \l_@@_name_str - last }
4524 { \@@_env: - last }
4525 }
4526 \endpgfpicture
4527 }

```

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.

```

4528 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4529 {

```

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

```
4530 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4531 \l_@@_initial_i_int = #1
4532 \l_@@_initial_j_int = #2
4533 \l_@@_final_i_int = #1
4534 \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.

```
4535 \let \l_@@_stop_loop_bool \c_false_bool
4536 \bool_do_until:Nn \l_@@_stop_loop_bool
4537 {
```

We test if we are still in the matrix.

```
4538 \advance \l_@@_final_i_int by #3
4539 \advance \l_@@_final_j_int by #4
4540 \let \l_@@_final_open_bool \c_false_bool
4541 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4542 \if_int_compare:w #3 = \c_one_int
4543 \let \l_@@_final_open_bool \c_true_bool
4544 \else:
4545 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4546 \let \l_@@_final_open_bool \c_true_bool
4547 \fi:
4548 \fi:
4549 \else:
4550 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4551 \if_int_compare:w #4 = -1
4552 \let \l_@@_final_open_bool \c_true_bool
4553 \fi:
4554 \else:
4555 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4556 \if_int_compare:w #4 = \c_one_int
4557 \let \l_@@_final_open_bool \c_true_bool
4558 \fi:
4559 \fi:
4560 \fi:
4561 \fi:
4562 \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.

```
4563 {
```

We do a step backwards.

```
4564 \advance \l_@@_final_i_int by - #3
4565 \advance \l_@@_final_j_int by - #4
4566 \let \l_@@_stop_loop_bool \c_true_bool
4567 }
```

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`.

```
4568 {
4569 \cs_if_exist:cTF
4570 {
4571 @@ _ dotted _
4572 \int_use:N \l_@@_final_i_int -
4573 \int_use:N \l_@@_final_j_int
4574 }
4575 {
4576 \advance \l_@@_final_i_int by - #3
4577 \advance \l_@@_final_j_int by - #4
4578 \let \l_@@_final_open_bool \c_true_bool
```

```

4579         \let \l_@@_stop_loop_bool \c_true_bool
4580     }
4581     {
4582         \cs_if_exist:cTF
4583         {
4584             pgf @ sh @ ns @ \@@_env:
4585             - \int_use:N \l_@@_final_i_int
4586             - \int_use:N \l_@@_final_j_int
4587         }
4588         { \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.

```

4589         {
4590             \cs_set_nopar:cpn
4591             {
4592                 @@ _ dotted _
4593                 \int_use:N \l_@@_final_i_int -
4594                 \int_use:N \l_@@_final_j_int
4595             }
4596             { }
4597         }
4598     }
4599 }
4600

```

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

```

4601     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4602     \l_tmpa_int = \l_@@_col_min_int
4603     \advance \l_tmpa_int by -1
4604     \bool_do_until:Nn \l_@@_stop_loop_bool
4605     {
4606         \advance \l_@@_initial_i_int by - #3
4607         \advance \l_@@_initial_j_int by - #4
4608         \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.

```

4609         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4610         \if_int_compare:w #3 = \c_one_int
4611         \let \l_@@_initial_open_bool \c_true_bool
4612     \else:

```

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

```

4613         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4614         \let \l_@@_initial_open_bool \c_true_bool
4615         \fi:
4616     \fi:
4617 \else:
4618     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4619     \if_int_compare:w #4 = \c_one_int
4620     \let \l_@@_initial_open_bool \c_true_bool
4621     \fi:
4622 \else:
4623     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4624     \if_int_compare:w #4 = -1
4625     \let \l_@@_initial_open_bool \c_true_bool

```

```

4626         \fi:
4627     \fi:
4628     \fi:
4629 \fi:
4630 \bool_if:NTF \l_@@_initial_open_bool
4631 {
4632     \advance \l_@@_initial_i_int by #3
4633     \advance \l_@@_initial_j_int by #4
4634     \let \l_@@_stop_loop_bool \c_true_bool
4635 }
4636 {
4637     \cs_if_exist:cTF
4638     {
4639         @@ _ dotted _
4640         \int_use:N \l_@@_initial_i_int -
4641         \int_use:N \l_@@_initial_j_int
4642     }
4643     {
4644         \advance \l_@@_initial_i_int by #3
4645         \advance \l_@@_initial_j_int by #4
4646         \let \l_@@_initial_open_bool \c_true_bool
4647         \let \l_@@_stop_loop_bool \c_true_bool
4648     }
4649     {
4650         \cs_if_exist:cTF
4651         {
4652             pgf @ sh @ ns @ \@@_env:
4653             - \int_use:N \l_@@_initial_i_int
4654             - \int_use:N \l_@@_initial_j_int
4655         }
4656         { \let \l_@@_stop_loop_bool \c_true_bool }
4657     }
4658     \cs_set_nopar:cpn
4659     {
4660         @@ _ dotted _
4661         \int_use:N \l_@@_initial_i_int -
4662         \int_use:N \l_@@_initial_j_int
4663     }
4664     { }
4665 }
4666 }
4667 }
4668 }

```

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.

```

4669 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4670 {
4671     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4672     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4673     { \int_use:N \l_@@_final_i_int }
4674     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4675     { }
4676 }
4677 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we known whether the extremities are closed or open) but before the analysis of

the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4678 \cs_new_protected:Npn \@@_open_shorten:
4679 {
4680   \bool_if:NT \l_@@_initial_open_bool
4681     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4682   \bool_if:NT \l_@@_final_open_bool
4683     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4684 }

```

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).

```

4685 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4686 {
4687   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4688   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4689   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4690   \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.

```

4691   \seq_if_empty:NTF \g_@@_submatrix_seq
4692     { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4693     {

```

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`.

```

4694       \seq_map_inline:Nn \g_@@_submatrix_seq
4695       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4696     }
4697 }

```

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

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

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4698 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4699 {
4700   \if_int_compare:w #3 > #1
4701   \else:
4702     \if_int_compare:w #1 > #5
4703     \else:
4704       \if_int_compare:w #4 > #2
4705       \else:

```

```

4706         \if_int_compare:w #2 > #6
4707         \else:
4708             \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4709             \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4710             \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4711             \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4712         \fi:
4713     \fi:
4714 \fi:
4715 \fi:
4716 }

4717 \cs_new_protected:Npn \@@_set_initial_coords:
4718 {
4719     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4720     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4721 }
4722 \cs_new_protected:Npn \@@_set_final_coords:
4723 {
4724     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4725     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4726 }
4727 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4728 {
4729     \pgfpointanchor
4730     {
4731         \@@_env:
4732         - \int_use:N \l_@@_initial_i_int
4733         - \int_use:N \l_@@_initial_j_int
4734     }
4735     { #1 }
4736     \@@_set_initial_coords:
4737 }
4738 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4739 {
4740     \pgfpointanchor
4741     {
4742         \@@_env:
4743         - \int_use:N \l_@@_final_i_int
4744         - \int_use:N \l_@@_final_j_int
4745     }
4746     { #1 }
4747     \@@_set_final_coords:
4748 }
4749 \cs_new_protected:Npn \@@_open_x_initial_dim:
4750 {
4751     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4752     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4753     {
4754         \cs_if_exist:cT
4755         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4756         {
4757             \pgfpointanchor
4758             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4759             { west }
4760             \dim_set:Nn \l_@@_x_initial_dim
4761             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4762         }
4763     }

```

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

```

4764     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4765     {

```

```

4766     \l_@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4767     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4768     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4769   }
4770 }
4771 \cs_new_protected:Npn \l_@@_open_x_final_dim:
4772 {
4773   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4774   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4775   {
4776     \cs_if_exist:cT
4777     { \pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4778     {
4779       \pgfpointanchor
4780       { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4781       { east }
4782       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4783       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4784     }
4785   }

```

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

```

4786   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4787   {
4788     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4789     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4790     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4791   }
4792 }

```

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.

```

4793 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4794 {
4795   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4796   \cs_if_free:cT { \l_@@_dotted_ #1 - #2 }
4797   {
4798     \l_@@_find_extremities:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4799   \bool_if:NT \g_@@_aux_found_bool
4800   {
4801     {
4802       \l_@@_open_shorten:
4803       \int_if_zero:nTF { #1 }
4804       { \color { nicematrix-first-row } }
4805     }

```

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.

```

4806       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4807       { \color { nicematrix-last-row } }
4808     }
4809     \keys_set:nn { nicematrix / xdots } { #3 }
4810     \l_@@_color:o \l_@@_xdots_color_tl
4811     \l_@@_actually_draw_Ldots:
4812   }
4813 }
4814 }
4815 }

```

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

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

The following function is also used by \Hdotsfor.

```

4816 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4817 {
4818   \bool_if:NTF \l_@@_initial_open_bool
4819   {
4820     \@@_open_x_initial_dim:
4821     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4822     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4823   }
4824   { \@@_set_initial_coords_from_anchor:n { base-east } }
4825   \bool_if:NTF \l_@@_final_open_bool
4826   {
4827     \@@_open_x_final_dim:
4828     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4829     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4830   }
4831   { \@@_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.

```

4832   \bool_lazy_all:nTF
4833   {
4834     \l_@@_initial_open_bool
4835     \l_@@_final_open_bool
4836     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4837   }
4838   {
4839     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4840     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4841   }

```

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 (?).

```

4842   {
4843     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4844     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4845   }
4846   \@@_draw_line:
4847 }

```

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.

```

4848 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4849 {
4850   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4851   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4852   {
4853     \@@_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.

```

4854     \bool_if:NT \g_@@_aux_found_bool
4855     {
4856     {
4857         \@@_open_shorten:
4858         \int_if_zero:nTF { #1 }
4859         { \color { nicematrix-first-row } }
4860         {

```

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.

```

4861         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4862         { \color { nicematrix-last-row } }
4863     }
4864     \keys_set:nn { nicematrix / xdots } { #3 }
4865     \@@_color:o \l_@@_xdots_color_tl
4866     \@@_actually_draw_Cdots:
4867 }
4868 }
4869 }
4870 }

```

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`.

```

4871 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4872 {
4873     \bool_if:NTF \l_@@_initial_open_bool
4874     \@@_open_x_initial_dim:
4875     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4876     \bool_if:NTF \l_@@_final_open_bool
4877     \@@_open_x_final_dim:
4878     { \@@_set_final_coords_from_anchor:n { mid-west } }
4879     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4880     {
4881         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4882         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4883         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4884         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4885         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4886     }
4887     {
4888         \bool_if:NT \l_@@_initial_open_bool
4889         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4890         \bool_if:NT \l_@@_final_open_bool
4891         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4892     }
4893     \@@_draw_line:
4894 }
4895 \cs_new_protected:Npn \@@_open_y_initial_dim:
4896 {
4897     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4898     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int

```

```

4899 {
4900   \cs_if_exist:cT
4901     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4902     {
4903       \pgfpointanchor
4904         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4905         { north }
4906       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4907       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4908     }
4909   }
4910   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4911   {
4912     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4913     \dim_set:Nn \l_@@_y_initial_dim
4914       {
4915         \fp_to_dim:n
4916         {
4917           \pgf@y
4918           + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4919         }
4920       }
4921   }
4922 }
4923 \cs_new_protected:Npn \@@_open_y_final_dim:
4924 {
4925   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4926   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4927   {
4928     \cs_if_exist:cT
4929       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4930       {
4931         \pgfpointanchor
4932           { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4933           { south }
4934         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4935         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4936       }
4937   }
4938   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4939   {
4940     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4941     \dim_set:Nn \l_@@_y_final_dim
4942       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4943   }
4944 }

```

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.

```

4945 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4946 {
4947   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4948   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4949   {
4950     \@@_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.

```

4951   \bool_if:NT \g_@@_aux_found_bool
4952   {
4953     {
4954       \@@_open_shorten:
4955       \int_if_zero:nTF { #2 }
4956       { \color { nicematrix-first-col } }

```

```

4957         {
4958             \int_compare:nNtT { #2 } = \l_@@_last_col_int
4959             { \color { nicematrix-last-col } }
4960         }
4961         \keys_set:nn { nicematrix / xdots } { #3 }
4962         \@@_color:o \l_@@_xdots_color_tl
4963         \bool_if:NTF \l_@@_Vbrace_bool
4964             \@@_actually_draw_Vbrace:
4965             \@@_actually_draw_Vdots:
4966     }
4967 }
4968 }
4969 }

```

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`.

```

4970 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4971 {
4972     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4973     \@@_actually_draw_Vdots_i:
4974     \@@_actually_draw_Vdots_ii:
4975     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4976     \@@_draw_line:
4977 }

```

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

```

4978 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4979 {
4980     \@@_open_y_initial_dim:
4981     \@@_open_y_final_dim:
4982     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4983     {
4984         \@@_qpoint:n { col - 1 }
4985         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4986         \dim_sub:Nn \l_@@_x_initial_dim
4987         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4988     }
4989     {
4990         \bool_lazy_and:nnTF
4991         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4992         { \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”.

```

4993     {
4994         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4995         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4996         \dim_add:Nn \l_@@_x_initial_dim
4997         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4998     }

```

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

```

4999      {
5000          \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5001          \dim_set_eq:NN \l_tmpa_dim \pgf@x
5002          \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
5003          \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
5004      }
5005  }
5006 }

```

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

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

5007 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
5008 {
5009     \bool_set_false:N \l_tmpa_bool
5010     \bool_if:NF \l_@@_initial_open_bool
5011     {
5012         \bool_if:NF \l_@@_final_open_bool
5013         {
5014             \@@_set_initial_coords_from_anchor:n { south-west }
5015             \@@_set_final_coords_from_anchor:n { north-west }
5016             \bool_set:Nn \l_tmpa_bool
5017                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
5018         }
5019     }

```

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

```

5020     \bool_if:NTF \l_@@_initial_open_bool
5021     {
5022         \@@_open_y_initial_dim:
5023         \@@_set_final_coords_from_anchor:n { north }
5024         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
5025     }
5026     {
5027         \@@_set_initial_coords_from_anchor:n { south }
5028         \bool_if:NTF \l_@@_final_open_bool
5029         \@@_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.

```

5030     {
5031         \@@_set_final_coords_from_anchor:n { north }
5032         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
5033         {
5034             \dim_set:Nn \l_@@_x_initial_dim
5035             {
5036                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
5037                     \l_@@_x_initial_dim \l_@@_x_final_dim
5038             }
5039         }
5040     }
5041 }
5042 }

```

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`.

```

5043 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
5044 {
5045   \bool_if:NTF \l_@@_initial_open_bool
5046     \@@_open_y_initial_dim:
5047     { \@@_set_initial_coords_from_anchor:n { south } }
5048   \bool_if:NTF \l_@@_final_open_bool
5049     \@@_open_y_final_dim:
5050     { \@@_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.

```

5051   \int_if_zero:nTF \l_@@_initial_j_int
5052   {
5053     \@@_qpoint:n { col - 1 }
5054     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5055     \dim_sub:Nn \l_@@_x_initial_dim
5056     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
5057   }

```

Elsewhere, the brace must be drawn left flush.

```

5058   {
5059     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
5060     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5061     \dim_add:Nn \l_@@_x_initial_dim
5062     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
5063   }

```

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

```

5064   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5065   \@@_draw_line:
5066 }

```

```

5067 \cs_new:Npn \@@_colsep:
5068 { \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.

```

5069 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
5070 {
5071   \@@_adjust_to_submatrix:nn { #1 } { #2 }
5072   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5073   {
5074     \@@_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.

```

5075   \bool_if:NT \g_@@_aux_found_bool
5076   {
5077     {
5078       \@@_open_shorten:
5079       \keys_set:nn { nicematrix / xdots } { #3 }

```

```

5080         \l_@@_xdots_color_tl
5081         \l_@@_actually_draw_Ddots:
5082     }
5083 }
5084 }
5085 }

```

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

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

```

5086 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
5087 {
5088     \bool_if:NTF \l_@@_initial_open_bool
5089     {
5090         \l_@@_open_y_initial_dim:
5091         \l_@@_open_x_initial_dim:
5092     }
5093     { \l_@@_set_initial_coords_from_anchor:n { south-east } }
5094     \bool_if:NTF \l_@@_final_open_bool
5095     {
5096         \l_@@_open_x_final_dim:
5097         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5098     }
5099     { \l_@@_set_final_coords_from_anchor:n { north-west } }

```

We have retrieved the coordinates in the usual way (they are stored in `\l_@@_x_initial_dim`, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

```

5100     \bool_if:NT \l_@@_parallelize_diags_bool
5101     {
5102         \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).

```

5103         \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.

```

5104     {
5105         \dim_gset:Nn \g_@@_delta_x_one_dim
5106         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5107         \dim_gset:Nn \g_@@_delta_y_one_dim
5108         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5109     }

```

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`.

```

5110     {
5111         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
5112         {
5113             \dim_set:Nn \l_@@_y_final_dim
5114             {
5115                 \l_@@_y_initial_dim +
5116                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5117                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
5118             }

```

```

5119         }
5120     }
5121 }
5122 \@@_draw_line:
5123 }

```

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.

```

5124 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
5125 {
5126     \@@_adjust_to_submatrix:nn { #1 } { #2 }
5127     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
5128     {
5129         \@@_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.

```

5130         \bool_if:NT \g_@@_aux_found_bool
5131         {
5132             {
5133                 \@@_open_shorten:
5134                 \keys_set:nn { nicematrix / xdots } { #3 }
5135                 \@@_color:o \l_@@_xdots_color_tl
5136                 \@@_actually_draw_Iddots:
5137             }
5138         }
5139     }
5140 }

```

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`.

```

5141 \cs_new_protected:Npn \@@_actually_draw_Iddots:
5142 {
5143     \bool_if:NTF \l_@@_initial_open_bool
5144     {
5145         \@@_open_y_initial_dim:
5146         \@@_open_x_initial_dim:
5147     }
5148     { \@@_set_initial_coords_from_anchor:n { south~west } }
5149     \bool_if:NTF \l_@@_final_open_bool
5150     {
5151         \@@_open_y_final_dim:
5152         \@@_open_x_final_dim:
5153     }
5154     { \@@_set_final_coords_from_anchor:n { north~east } }
5155     \bool_if:NT \l_@@_parallelize_diags_bool
5156     {
5157         \int_gincr:N \g_@@_iddots_int
5158         \int_compare:nNnTF \g_@@_iddots_int = 1
5159         {
5160             \dim_gset:Nn \g_@@_delta_x_two_dim

```

```

5161         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
5162         \dim_gset:Nn \g_@@_delta_y_two_dim
5163         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
5164     }
5165     {
5166         \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
5167         {
5168             \dim_set:Nn \l_@@_y_final_dim
5169             {
5170                 \l_@@_y_initial_dim +
5171                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5172                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
5173             }
5174         }
5175     }
5176 }
5177 \@@_draw_line:
5178 }

```

17 The actual instructions for drawing the dotted lines with TikZ

The command `\@@_draw_line:` has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

5179 \cs_new_protected:Npn \@@_draw_line:
5180 {
5181     \pgfrememberpicturepositiononpagetrue
5182     \pgf@relevantforpicturesizefalse
5183     \bool_lazy_or:nnTF
5184         \l_@@_dotted_bool
5185         { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
5186         \@@_draw_standard_dotted_line:
5187         \@@_draw_unstandard_dotted_line:
5188 }

```

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.

```

5189 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
5190 {
5191     \begin { scope }
5192     \@@_draw_unstandard_dotted_line:o
5193         { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
5194 }

```

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.

```

5195 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
5196 {
5197   \@@_draw_unstandard_dotted_line:nnnn #1
5198   { #1 }
5199   \l_@@_xdots_up_tl
5200   \l_@@_xdots_down_tl
5201   \l_@@_xdots_middle_tl
5202 }
5203 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

5204 \AtBeginDocument
5205 {
5206   \IfPackageLoadedT { tikz }
5207   {
5208     \tikzset
5209     {
5210       @@_node_above / .style = { sloped , above } ,
5211       @@_node_below / .style = { sloped , below } ,
5212       @@_node_middle / .style =
5213       {
5214         sloped ,
5215         inner~sep = \c_@@_innersep_middle_dim
5216       }
5217     }
5218   }
5219 }

5220 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
5221 {

```

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.

```

5222 \dim_zero_new:N \l_@@_l_dim
5223 \dim_set:Nn \l_@@_l_dim
5224 {
5225   \fp_to_dim:n
5226   {
5227     sqrt
5228     (
5229       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5230       +
5231       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5232     )
5233   }
5234 }

```

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.

```

5235 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5236 {
5237   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5238   \@@_draw_unstandard_dotted_line_i:
5239 }

```

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

```

5240 \bool_if:NT \l_@@_xdots_h_labels_bool
5241 {
5242   \tikzset
5243   {
5244     @@_node_above / .style = { auto = left } ,
5245     @@_node_below / .style = { auto = right } ,
5246     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
5247   }
5248 }
5249 \tl_if_empty:nF { #4 }
5250 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
5251 \dim_zero:N \l_tmpa_dim
5252 \dim_zero:N \l_tmpb_dim
5253 \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
5254 {

```

We test whether the brace is vertical or horizontal.

```

5255 \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5256 { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
5257 { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
5258 }
5259 {
5260   \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
5261   {
5262     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
5263     { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
5264     { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
5265   }
5266 }
5267 \use:e
5268 {
5269   \exp_not:N \begin { scope }
5270   [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
5271   }
5272 \draw
5273 [ #1 ]
5274 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
5275 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
5276 node [ @@_node_below ] { $ \scriptstyle #3 $ }
5277 node [ @@_node_above ] { $ \scriptstyle #2 $ }
5278 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
5279 \end { scope }
5280 \end { scope }
5281 }
5282 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
5283 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
5284 {
5285   \dim_set:Nn \l_tmpa_dim
5286   {
5287     \l_@@_x_initial_dim
5288     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5289     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5290   }
5291   \dim_set:Nn \l_tmpb_dim
5292   {
5293     \l_@@_y_initial_dim
5294     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5295     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5296   }
5297   \dim_set:Nn \l_@@_tmpc_dim
5298   {
5299     \l_@@_x_final_dim

```

```

5300     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5301     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5302   }
5303   \dim_set:Nn \l_@@_tmpd_dim
5304   {
5305     \l_@@_y_final_dim
5306     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5307     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5308   }
5309   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5310   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5311   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5312   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5313 }

```

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

```

5314 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5315 {
5316 {

```

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.

```

5317   \dim_zero_new:N \l_@@_l_dim
5318   \dim_set:Nn \l_@@_l_dim
5319   {
5320     \fp_to_dim:n
5321     {
5322       sqrt
5323       (
5324         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5325         +
5326         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5327       )
5328     }
5329   }

```

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.

```

5330   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5331   {
5332     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5333     \@@_draw_standard_dotted_line_i:
5334   }
5335 }
5336 \bool_lazy_all:nF
5337 {
5338   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5339   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5340   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5341 }
5342 \@@_labels_standard_dotted_line:
5343 }
5344 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5345 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5346 {

```

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

```

5347   \int_set:Nn \l_tmpa_int
5348   {

```

```

5349 \dim_ratio:nn
5350 {
5351   \l_@@_l_dim
5352   - \l_@@_xdots_shorten_start_dim
5353   - \l_@@_xdots_shorten_end_dim
5354 }
5355 \l_@@_xdots_inter_dim
5356 }

```

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

```

5357 \dim_set:Nn \l_tmpa_dim
5358 {
5359   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5360   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5361 }
5362 \dim_set:Nn \l_tmpb_dim
5363 {
5364   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5365   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5366 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5367 \dim_gadd:Nn \l_@@_x_initial_dim
5368 {
5369   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5370   \dim_ratio:nn
5371   {
5372     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5373     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5374   }
5375   { 2 \l_@@_l_dim }
5376 }
5377 \dim_gadd:Nn \l_@@_y_initial_dim
5378 {
5379   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5380   \dim_ratio:nn
5381   {
5382     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5383     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5384   }
5385   { 2 \l_@@_l_dim }
5386 }
5387 \pgf@relevantforpicturesizefalse
5388 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5389 {
5390   \pgfpathcircle
5391   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5392   \l_@@_xdots_radius_dim
5393   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5394   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5395 }
5396 \pgfusepathqfill
5397 }

```

```

5398 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5399 {
5400   \pgfscope
5401   \pgftransformshift
5402   {
5403     \pgfpointlineatime { 0.5 }
5404     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }

```

```

5405         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5406     }
5407     \fp_set:Nn \l_tmpa_fp
5408     {
5409         atand
5410         (
5411             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5412             \l_@@_x_final_dim - \l_@@_x_initial_dim
5413         )
5414     }
5415     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5416     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5417     \tl_if_empty:NF \l_@@_xdots_middle_tl
5418     {
5419         \begin { pgfscope }
5420         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5421         \pgfnode
5422         { rectangle }
5423         { center }
5424         {
5425             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5426             {
5427                 $ % $
5428                 \scriptstyle \l_@@_xdots_middle_tl
5429                 $ % $
5430             }
5431         }
5432         { }
5433         {
5434             \pgfsetfillcolor { white }
5435             \pgfusepath { fill }
5436         }
5437         \end { pgfscope }
5438     }
5439     \tl_if_empty:NF \l_@@_xdots_up_tl
5440     {
5441         \pgfnode
5442         { rectangle }
5443         { south }
5444         {
5445             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5446             {
5447                 $ % $
5448                 \scriptstyle \l_@@_xdots_up_tl
5449                 $ % $
5450             }
5451         }
5452         { }
5453         { \pgfusepath { } }
5454     }
5455     \tl_if_empty:NF \l_@@_xdots_down_tl
5456     {
5457         \pgfnode
5458         { rectangle }
5459         { north }
5460         {
5461             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5462             {
5463                 $ % $
5464                 \scriptstyle \l_@@_xdots_down_tl
5465                 $ % $
5466             }
5467         }
5468     }

```

```

5468         { }
5469         { \pgfusepath { } }
5470     }
5471 \endpgfscope
5472 }

```

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.

```

5473 \AtBeginDocument
5474 {

```

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

```

5475     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5476     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5477     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5478     {
5479         \int_if_zero:nTF \c@jCol
5480         { \@@_error:nn { in~first~col } { \Ldots } }
5481         {
5482             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5483             { \@@_error:nn { in~last~col } { \Ldots } }
5484             {
5485                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5486                 { #1 , down = #2 , up = #3 , middle = #4 }
5487             }
5488         }
5489         \bool_if:NF \l_@@_nullify_dots_bool
5490         { \phantom { \ensuremath { \@@_old_ldots: } } }
5491         \bool_gset_true:N \g_@@_empty_cell_bool
5492     }

5493     \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5494     \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5495     {
5496         \int_if_zero:nTF \c@jCol
5497         { \@@_error:nn { in~first~col } { \Cdots } }
5498         {
5499             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5500             { \@@_error:nn { in~last~col } { \Cdots } }
5501             {
5502                 \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5503                 { #1 , down = #2 , up = #3 , middle = #4 }
5504             }
5505         }
5506         \bool_if:NF \l_@@_nullify_dots_bool
5507         { \phantom { \ensuremath { \@@_old_cdots: } } }
5508         \bool_gset_true:N \g_@@_empty_cell_bool
5509     }

```

```

5510 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5511 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5512 {
5513   \int_if_zero:nTF \c@iRow
5514   { \@@_error:nn { in~first~row } { \Vdots } }
5515   {
5516     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5517     { \@@_error:nn { in~last~row } { \Vdots } }
5518     {
5519       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5520       { #1 , down = #2 , up = #3 , middle = #4 }
5521     }
5522   }
5523   \bool_if:NF \l_@@_nullify_dots_bool
5524   { \phantom { \ensuremath { \@@_old_vdots: } } }
5525   \bool_gset_true:N \g_@@_empty_cell_bool
5526 }

5527 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5528 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5529 {
5530   \int_case:nnF \c@iRow
5531   {
5532     0 { \@@_error:nn { in~first~row } { \Ddots } }
5533     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5534   }
5535   {
5536     \int_case:nnF \c@jCol
5537     {
5538       0 { \@@_error:nn { in~first~col } { \Ddots } }
5539       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5540     }
5541     {
5542       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5543       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5544       { #1 , down = #2 , up = #3 , middle = #4 }
5545     }
5546   }
5547 }
5548 \bool_if:NF \l_@@_nullify_dots_bool
5549 { \phantom { \ensuremath { \@@_old_ddots: } } }
5550 \bool_gset_true:N \g_@@_empty_cell_bool
5551 }

5552 \cs_new_protected:Npn \@@_Iddots:
5553 { \@@_collect_options:n { \@@_Iddots_i } }
5554 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5555 {
5556   \int_case:nnF \c@iRow
5557   {
5558     0 { \@@_error:nn { in~first~row } { \Iddots } }
5559     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5560   }
5561   {
5562     \int_case:nnF \c@jCol
5563     {
5564       0 { \@@_error:nn { in~first~col } { \Iddots } }
5565       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5566     }
5567     {
5568       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5569       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }

```

```

5570         { #1 , down = #2 , up = #3 , middle = #4 }
5571     }
5572 }
5573 \bool_if:NF \l_@@_nullify_dots_bool
5574 { \phantom { \ensuremath { \@@_old_iddots: } } }
5575 \bool_gset_true:N \g_@@_empty_cell_bool
5576 }
5577 }

```

End of the `\AddToHook`.

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

```

5578 \keys_define:nn { nicematrix / Ddots }
5579 {
5580     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5581     draw-first .value_forbidden:n = true ,
5582 }

```

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

```

5583 \cs_new_protected:Npn \@@_Hspace:
5584 {
5585     \bool_gset_true:N \g_@@_empty_cell_bool
5586     \hspace
5587 }

```

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`.

```

5588 \cs_new:Npn \@@_Hdotsfor:
5589 {
5590     \bool_lazy_and:nnTF
5591     { \int_if_zero_p:n \c@jCol }
5592     { \int_if_zero_p:n \l_@@_first_col_int }
5593     {
5594         \bool_if:NTF \g_@@_after_col_zero_bool
5595         {
5596             \multicolumn { 1 } { c } { }
5597             \@@_Hdotsfor_i:
5598         }
5599         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5600     }
5601     {
5602         \multicolumn { 1 } { c } { }
5603         \@@_Hdotsfor_i:
5604     }
5605 }

```

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:`).

```

5606 \AtBeginDocument
5607 {

```

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

```

5608     \cs_new_protected:Npn \@@_Hdotsfor_i:
5609     { \@@_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`).

```

5610     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5611     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl

```

```

5612 {
5613   \tl_gput_right:N\g_@@_HVDotsfor_lines_tl
5614   {
5615     \@@_Hdotsfor:nnnn
5616     { \int_use:N \c@iRow }
5617     { \int_use:N \c@jCol }
5618     { #2 }
5619     {
5620       #1 , #3 ,
5621       down = \exp_not:n { #4 } ,
5622       up = \exp_not:n { #5 } ,
5623       middle = \exp_not:n { #6 }
5624     }
5625   }
5626   \prg_replicate:nn { #2 - 1 }
5627   {
5628     &
5629     \multicolumn { 1 } { c } { }
5630     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5631   }
5632 }
5633 }

```

```

5634 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5635 {
5636   \bool_set_false:N \l_@@_initial_open_bool
5637   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5638   \int_set:Nn \l_@@_initial_i_int { #1 }
5639   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5640   \int_compare:nNnTF { #2 } = 1
5641   {
5642     \int_set:Nn \l_@@_initial_j_int 1
5643     \bool_set_true:N \l_@@_initial_open_bool
5644   }
5645   {
5646     \cs_if_exist:cTF
5647     {
5648       pgf @ sh @ ns @ \@@_env:
5649       - \int_use:N \l_@@_initial_i_int
5650       - \int_eval:n { #2 - 1 }
5651     }
5652     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5653     {
5654       \int_set:Nn \l_@@_initial_j_int { #2 }
5655       \bool_set_true:N \l_@@_initial_open_bool
5656     }
5657   }
5658   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5659   {
5660     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5661     \bool_set_true:N \l_@@_final_open_bool
5662   }
5663   {
5664     \cs_if_exist:cTF
5665     {
5666       pgf @ sh @ ns @ \@@_env:
5667       - \int_use:N \l_@@_final_i_int
5668       - \int_eval:n { #2 + #3 }
5669     }
5670     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }

```

```

5671     {
5672         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5673         \bool_set_true:N \l_@@_final_open_bool
5674     }
5675 }
5676 \bool_if:NT \g_@@_aux_found_bool
5677 {
5678     {
5679         \@@_open_shorten:
5680         \int_if_zero:nTF { #1 }
5681         { \color { nicematrix-first-row } }
5682         {
5683             \int_compare:nNtT { #1 } = \g_@@_row_total_int
5684             { \color { nicematrix-last-row } }
5685         }
5686         \keys_set:nn { nicematrix / xdots } { #4 }
5687         \@@_color:o \l_@@_xdots_color_tl
5688         \@@_actually_draw_Ldots:
5689     }
5690 }

```

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).

```

5691     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5692     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5693 }

```

```

5694 \AtBeginDocument
5695 {
5696     \cs_new_protected:Npn \@@_Vdotsfor:
5697     { \@@_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`).

```

5698     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } { } } }
5699     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5700     {
5701         \bool_gset_true:N \g_@@_empty_cell_bool
5702         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5703         {
5704             \@@_Vdotsfor:nnnn
5705             { \int_use:N \c@iRow }
5706             { \int_use:N \c@jCol }
5707             { #2 }
5708             {
5709                 #1 , #3 ,
5710                 down = \exp_not:n { #4 } ,
5711                 up = \exp_not:n { #5 } ,
5712                 middle = \exp_not:n { #6 }
5713             }
5714         }
5715     }
5716 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5717 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5718 {
5719     \bool_set_false:N \l_@@_initial_open_bool
5720     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5721 \int_set:Nn \l_@@_initial_j_int { #2 }
5722 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5723 \int_compare:nNnTF { #1 } = 1
5724 {
5725   \int_set:Nn \l_@@_initial_i_int 1
5726   \bool_set_true:N \l_@@_initial_open_bool
5727 }
5728 {
5729   \cs_if_exist:cTF
5730   {
5731     pgf @ sh @ ns @ \@@_env:
5732     - \int_eval:n { #1 - 1 }
5733     - \int_use:N \l_@@_initial_j_int
5734   }
5735   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5736   {
5737     \int_set:Nn \l_@@_initial_i_int { #1 }
5738     \bool_set_true:N \l_@@_initial_open_bool
5739   }
5740 }
5741 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5742 {
5743   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5744   \bool_set_true:N \l_@@_final_open_bool
5745 }
5746 {
5747   \cs_if_exist:cTF
5748   {
5749     pgf @ sh @ ns @ \@@_env:
5750     - \int_eval:n { #1 + #3 }
5751     - \int_use:N \l_@@_final_j_int
5752   }
5753   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5754   {
5755     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5756     \bool_set_true:N \l_@@_final_open_bool
5757   }
5758 }
5759 \bool_if:NT \g_@@_aux_found_bool
5760 {
5761   {
5762     \@@_open_shorten:
5763     \int_if_zero:nTF { #2 }
5764     { \color { nicematrix-first-col } }
5765     {
5766       \int_compare:nNnT { #2 } = \g_@@_col_total_int
5767       { \color { nicematrix-last-col } }
5768     }
5769     \keys_set:nn { nicematrix / xdots } { #4 }
5770     \@@_color:o \l_@@_xdots_color_tl
5771     \bool_if:NTF \l_@@_Vbrace_bool
5772     \@@_actually_draw_Vbrace:
5773     \@@_actually_draw_Vdots:
5774   }
5775 }
```

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).

```
5776 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
```

```

5777     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5778 }

```

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

```

5779 \NewDocumentCommand \@@_rotate: { 0 { } }
5780 {
5781   \bool_gset_true:N \g_@@_rotate_bool
5782   \keys_set:nn { nicematrix / rotate } { #1 }
5783   \ignorespaces
5784 }

```

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

```

5785 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5786 \keys_define:nn { nicematrix / rotate }
5787 {
5788   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5789   c .value_forbidden:n = true ,
5790   -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5791   -90 .value_forbidden:n = true ,
5792   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5793 }

```

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

```

5794 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5795 {
5796   \tl_if_empty:nTF { #2 }
5797   { #1 }
5798   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5799 }
5800 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5801 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

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

```

5802 \AtBeginDocument
5803 {

```

¹⁵Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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

```

5804 \tl_set_rescan:Nnn \l_tmpa_tl { }
5805 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5806 \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5807 {
5808   {
5809     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5810     \@@_color:o \l_@@_xdots_color_tl
5811     \use:e
5812     {
5813       \@@_line_i:nn
5814       { \@@_double_int_eval:n #2 - \q_stop }
5815       { \@@_double_int_eval:n #3 - \q_stop }
5816     }
5817   }
5818 }
5819 }

5820 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5821 {
5822   \bool_set_false:N \l_@@_initial_open_bool
5823   \bool_set_false:N \l_@@_final_open_bool
5824   \bool_lazy_or:nnTF
5825   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5826   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5827   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5828 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5829 }

5830 \AtBeginDocument
5831 {
5832   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5833   {

```

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:`.

```

5834 \c_@@_pgfortikzpicture_tl
5835 \@@_draw_line_iii:nn { #1 } { #2 }
5836 \c_@@_endpgfortikzpicture_tl
5837 }
5838 }

```

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

```

5839 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5840 {
5841   \pgfrememberpicturepositiononpagetrue
5842   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5843   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5844   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5845   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5846   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5847   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5848   \@@_draw_line:
5849 }

```

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.

```
5850 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNt \c@iRow < }
5851 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNf \c@jCol < }
```

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

```
5852 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5853 {
5854   \tl_gput_right:Ne \g_@@_row_style_tl
5855   {
```

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

```
5856   \exp_not:N
5857   \@@_if_row_less_than:nn
5858   { \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`).

```
5859   {
5860     \exp_not:N
5861     \@@_if_col_greater_than:nn
5862     { \int_use:N \c@jCol }
5863     { \exp_not:n { #1 } \scan_stop: }
5864   }
5865 }
5866 }
5867 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5868 \keys_define:nn { nicematrix / RowStyle }
5869 {
5870   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5871   cell-space-top-limit+ .code:n =
5872     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5873   cell-space-top-limit+ .value_required:n = true ,
5874   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5875   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5876   cell-space-bottom-limit+ .code:n =
5877     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5878   cell-space-bottom-limit+ .value_required:n = true ,
5879   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5880   cell-space-limits .meta:n =
5881   {
5882     cell-space-top-limit = #1 ,
5883     cell-space-bottom-limit = #1 ,
5884   } ,
5885   cell-space-limits+ .meta:n =
5886   {
5887     cell-space-top-limit += #1 ,
5888     cell-space-bottom-limit += #1 ,
```

```

5889     } ,
5890     cell-space-limits+.meta:n = { cell-space-limits+ = #1 } ,
5891     color .tl_set:N = \l_@@_color_tl ,
5892     color .value_required:n = true ,
5893     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5894     nb-rows .code:n =
5895       \str_if_eq:eeTF { #1 } { * }
5896       { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5897       { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5898     nb-rows .value_required:n = true ,
5899     fill .tl_set:N = \l_@@_fill_tl ,
5900     fill .value_required:n = true ,

```

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

```

5901     opacity .tl_set:N = \l_@@_opacity_tl ,
5902     opacity .value_required:n = true ,
5903     rowcolor .tl_set:N = \l_@@_fill_tl ,
5904     rowcolor .value_required:n = true ,
5905     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5906     rounded-corners .default:n = 4 pt ,
5907     unknown .code:n =
5908       \@@_unknown_key:nn
5909       { nicematrix / RowStyle }
5910       { Unknown-key-for-RowStyle }
5911   }

```

```

5912 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5913 {
5914   \group_begin:
5915   \tl_clear:N \l_@@_fill_tl
5916   \tl_clear:N \l_@@_opacity_tl
5917   \tl_clear:N \l_@@_color_tl
5918   \int_set:Nn \l_@@_key_nb_rows_int 1
5919   \dim_zero:N \l_@@_rounded_corners_dim
5920   \dim_zero:N \l_tmpa_dim
5921   \dim_zero:N \l_tmpb_dim
5922   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5923   \tl_if_empty:NF \l_@@_fill_tl
5924   {
5925     \@@_add_opacity_to_fill:
5926     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5927     {

```

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

```

5928       \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5929       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5930       {
5931         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5932         - *
5933       }
5934       { \dim_use:N \l_@@_rounded_corners_dim }
5935     }
5936   }
5937   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5938   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5939   {
5940     \@@_put_in_row_style:e
5941     {
5942       \@@_put_in_cell_after_hook:n
5943       {

```

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

```

5944         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5945         { \dim_use:N \l_tmpa_dim }
5946     }
5947 }
5948 }

```

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

```

5949 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5950 {
5951     \@@_put_in_row_style:e
5952     {
5953         \@@_put_in_cell_after_hook:n
5954         {
5955             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5956             { \dim_use:N \l_tmpb_dim }
5957         }
5958     }
5959 }

```

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

```

5960 \tl_if_empty:NF \l_@@_color_tl
5961 {
5962     \@@_put_in_row_style:e
5963     {
5964         \mode_leave_vertical:
5965         \@@_color:n { \l_@@_color_tl }
5966     }
5967 }

```

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

```

5968 \bool_if:NT \l_@@_bold_row_style_bool
5969 {
5970     \@@_put_in_row_style:n
5971     {
5972         \exp_not:n
5973         {
5974             \if_mode_math:
5975             $ % $
5976             \bfseries \boldmath
5977             $ % $
5978             \else:
5979             \bfseries \boldmath
5980             \fi:
5981         }
5982     }
5983 }
5984 \group_end:
5985 \g_@@_row_style_tl
5986 \ignorespaces
5987 }

```

The following commande must *not* be protected.

```

5988 \cs_new:Npn \@@_rounded_from_row:n #1
5989 {
5990     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5991     { \int_eval:n { #1 } - 1 }
5992     {
5993         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5994         - \exp_not:n { \int_use:N \c@jCol }
5995     }
5996     { \dim_use:N \l_@@_rounded_corners_dim }
5997 }

```

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).

```
5998 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5999 {
```

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.

```
6000 \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`.

```
6001 \str_if_in:nnF { #1 } { !! }
6002 {
6003 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
6004 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
6005 }
6006 \int_if_zero:nTF \l_tmpa_int
```

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

```
6007 {
6008 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
6009 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
6010 }
```

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

```
6011 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
6012 }
6013 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
6014 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
6015 {
6016 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
6017 {
```

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

```
6018 \group_begin:
6019 \pgfsetcornersarced
6020 { \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.

```

6021     \bool_if:NTF \l_@@_hvlines_bool
6022     {
6023         \pgfpathrectanglecorners
6024         {
6025             \pgfpointadd
6026             { \@@_qpoint:n { row-1 } }
6027             { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
6028         }
6029         {
6030             \pgfpointadd
6031             {
6032                 \@@_qpoint:n
6033                 { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6034             }
6035             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
6036         }
6037     }
6038     {
6039         \pgfpathrectanglecorners
6040         { \@@_qpoint:n { row-1 } }
6041         {
6042             \pgfpointadd
6043             {
6044                 \@@_qpoint:n
6045                 { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
6046             }
6047             { \pgfpoint \c_zero_dim \arrayrulewidth }
6048         }
6049     }
6050     \pgfusepath { clip }
6051     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

6052     }
6053 }

```

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

```

6054 \cs_new_protected:Npn \@@_actually_color:
6055 {
6056     \pgfpicture
6057     \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.

```

6058     \@@_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`).

```

6059     \seq_map_indexed_inline:Nn \g_@@_colors_seq
6060     {
6061         \int_compare:nNnTF { ##1 } = 1
6062         {
6063             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
6064             \use:c { g_@@_color _ 1 _tl }
6065             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
6066         }
6067     }

```

```

6068         \pgfscope
6069         \@@_color_opacity: ##2
6070         \use:c { g_@@_color _ ##1 _tl }
6071         \tl_gclear:c { g_@@_color _ ##1 _tl }
6072         \pgfusepath { fill }
6073         \endpgfscope
6074     }
6075 }
6076 \endpgfpicture
6077 }

```

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

```

6078 \cs_new_protected:Npn \@@_color_opacity:
6079 {
6080     \peek_meaning:NTF [
6081         \@@_color_opacity:w
6082         { \@@_color_opacity:w [ ] }
6083     }

```

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.

```

6084 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
6085 {
6086     \tl_clear:N \l_tmpa_tl
6087     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

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

```

6088     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
6089     \tl_if_empty:NTF \l_tmpb_tl
6090         \declaredcolor
6091         { \use:e { \exp_not:N \undeclaredcolor [ \l_tmpb_tl ] } }
6092 }

```

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

```

6093 \keys_define:nn { nicematrix / color-opacity }
6094 {
6095     opacity .tl_set:N          = \l_tmpa_tl ,
6096     opacity .value_required:n = true
6097 }

```

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

```

6098 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
6099 {
6100     \def \l_@@_rows_tl { #1 }
6101     \def \l_@@_cols_tl { #2 }
6102     \@@_cartesian_path:
6103 }

```

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

```

6104 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
6105 {
6106     \tl_if_blank:nF { #2 }
6107     {
6108         \@@_add_to_colors_seq:en
6109         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6110         { \@@_cartesian_color:nn { #3 } { - } }
6111     }
6112 }

```

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

```
6113 \NewDocumentCommand \@@_columncolor { 0 { } m m }
6114 {
6115   \tl_if_blank:nF { #2 }
6116   {
6117     \@@_add_to_colors_seq:en
6118     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6119     { \@@_cartesian_color:nn { - } { #3 } }
6120   }
6121 }
```

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

```
6122 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
6123 {
6124   \tl_if_blank:nF { #2 }
6125   {
6126     \@@_add_to_colors_seq:en
6127     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6128     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
6129   }
6130 }
```

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

```
6131 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
6132 {
6133   \tl_if_blank:nF { #2 }
6134   {
6135     \@@_add_to_colors_seq:en
6136     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
6137     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
6138   }
6139 }
```

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

```
6140 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
6141 {
6142   \@@_cut_on_hyphen:w #1 \q_stop
6143   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6144   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6145   \@@_cut_on_hyphen:w #2 \q_stop
6146   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
6147   \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`.

```
6148 \@@_cartesian_path:n { #3 }
6149 }
```

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

```
6150 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
6151 {
6152   \clist_map_inline:nn { #3 }
6153   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
6154 }
```

```
6155 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
6156 {
6157   \int_step_inline:nn \c@iRow
6158   {
6159     \int_step_inline:nn \c@jCol
```

```

6160     {
6161         \int_if_even:nTF { ####1 + ##1 }
6162         { \@@_cellcolor [ #1 ] { #2 } }
6163         { \@@_cellcolor [ #1 ] { #3 } }
6164     { ##1 - ####1 }
6165     }
6166 }
6167 }

```

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”.

```

6168 \NewDocumentCommand \@@_arraycolor { 0 { } m }
6169 {
6170     \@@_rectanglecolor [ #1 ] { #2 }
6171     { 1 - 1 }
6172     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6173 }

6174 \keys_define:nn { nicematrix / rowcolors }
6175 {
6176     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
6177     cols .tl_set:N = \l_@@_cols_tl ,
6178     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
6179     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
6180 }

```

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*.

```

6181 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
6182 {

```

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

```

6183     \seq_clear_new:N \l_@@_colors_seq
6184     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
6185     \tl_clear_new:N \l_@@_cols_tl
6186     \tl_set:Nn \l_@@_cols_tl { - }
6187     \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).

```

6188     \int_zero_new:N \l_@@_color_int
6189     \int_set:Nn \l_@@_color_int 1
6190     \bool_if:NT \l_@@_respect_blocks_bool
6191     {

```

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`).

```

6192         \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
6193         { \@@_not_in_exterior_p:nnnnn ##1 }
6194     }

```

#2 is the list of intervals of rows.

```

6195     \clist_map_inline:nn { #2 }
6196     {
6197         \tl_set:Nn \l_tmpa_tl { ##1 }
6198         \tl_if_in:NnTF \l_tmpa_tl { - }
6199         { \@@_cut_on_hyphen:w ##1 \q_stop }
6200         { \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.

```

6201     \int_set:Nn \l_tmpa_int \l_tmpa_tl
6202     \int_set:Nn \l_@@_color_int
6203     { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
6204     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
6205     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
6206     {

```

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

```

6207         \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).

```

6208         \bool_if:NT \l_@@_respect_blocks_bool
6209         {
6210             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
6211             { \@@_intersect_our_row_p:nnnnn ###1 }
6212             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

6213         }
6214         \tl_set:Ne \l_@@_rows_tl
6215         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

6216         \tl_set:Ne \l_@@_color_tl
6217         {
6218             \@@_color_index:n
6219             {
6220                 \int_mod:nn
6221                 { \l_@@_color_int - 1 }
6222                 { \seq_count:N \l_@@_colors_seq }
6223                 + 1
6224             }
6225         }
6226         \tl_if_empty:NF \l_@@_color_tl
6227         {
6228             \use:e
6229             {
6230                 \@@_add_to_colors_seq:nn
6231                 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
6232                 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
6233             }
6234         }
6235         \int_incr:N \l_@@_color_int
6236         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
6237     }
6238 }
6239 }

```

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.

```

6240 \cs_new:Npn \@@_color_index:n #1
6241 {

```

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

```

6242     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
6243     { \@@_color_index:n { #1 - 1 } }
6244     { \seq_item:Nn \l_@@_colors_seq { #1 } }
6245 }

```

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

```

6246 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
6247 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

6248 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
6249 {
6250   \int_compare:nNt { #3 } > \l_tmpb_int
6251   { \int_set:Nn \l_tmpb_int { #3 } }
6252 }

```

```

6253 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
6254 {
6255   \int_if_zero:nTF { #4 }
6256   \prg_return_false:
6257   {
6258     \int_compare:nNtTF { #2 } > \c@jCol
6259     \prg_return_false:
6260     \prg_return_true:
6261   }
6262 }

```

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

```

6263 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
6264 {
6265   \int_compare:nNtTF { #1 } > \l_tmpa_int
6266   \prg_return_false:
6267   {
6268     \int_compare:nNtTF \l_tmpa_int > { #3 }
6269     \prg_return_false:
6270     \prg_return_true:
6271   }
6272 }

```

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).

```

6273 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
6274 {
6275   \dim_compare:nNtTF { #1 } = \c_zero_dim
6276   {
6277     \bool_if:NTF \l_@@_nocolor_used_bool
6278     { \@@_cartesian_path_normal_ii: }
6279     {

```

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.

```

6280       \clist_if_empty:NTF \l_@@_corners_cells_clist
6281       { \@@_cartesian_path_normal_i:n { #1 } }
6282       { \@@_cartesian_path_normal_ii: }
6283     }
6284   }
6285   { \@@_cartesian_path_normal_i:n { #1 } }
6286 }

```

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.

```
6287 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
6288 {
6289   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```

We begin the loop over the columns.

```
6290 \clist_map_inline:Nn \l_@@_cols_tl
6291 {
```

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

```
6292   \def \l_tmpa_tl { ##1 }
6293   \tl_if_in:NnTF \l_tmpa_tl { - }
6294     { \@@_cut_on_hyphen:w ##1 \q_stop }
6295     { \def \l_tmpb_tl { ##1 } }
6296   \tl_if_empty:NNTF \l_tmpa_tl
6297     { \def \l_tmpa_tl { 1 } }
6298     {
6299       \str_if_eq:eeT \l_tmpa_tl { * }
6300       { \def \l_tmpa_tl { 1 } }
6301     }
6302   \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6303     { \@@_error:n { Invalid~col~number } }
6304   \tl_if_empty:NNTF \l_tmpb_tl
6305     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6306     {
6307       \str_if_eq:eeT \l_tmpb_tl { * }
6308       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6309     }
6310   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6311     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```

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

```
6312   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6313   \@@_qpoint:n { col - \l_tmpa_tl }
6314   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6315     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6316     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6317   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6318   \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.

```
6319   \clist_map_inline:Nn \l_@@_rows_tl
6320   {
6321     \def \l_tmpa_tl { #####1 }
6322     \tl_if_in:NnTF \l_tmpa_tl { - }
6323       { \@@_cut_on_hyphen:w #####1 \q_stop }
6324       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6325     \tl_if_empty:NNTF \l_tmpa_tl
6326       { \def \l_tmpa_tl { 1 } }
6327       {
6328         \str_if_eq:eeT \l_tmpa_tl { * }
6329         { \def \l_tmpa_tl { 1 } }
6330       }
6331     \tl_if_empty:NNTF \l_tmpb_tl
6332       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6333       {
6334         \str_if_eq:eeT \l_tmpb_tl { * }
6335         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6336       }
6337     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6338       { \@@_error:n { Invalid~row~number } }
6339     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6340       { \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`.

```

6341         \cs_if_exist:cF
6342         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6343         {
6344             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6345             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6346             \@@_qpoint:n { row - \l_tmpa_tl }
6347             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6348             \pgfpathrectanglecorners
6349             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6350             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6351         }
6352     }
6353 }
6354 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6355 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6356 {
6357     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6358     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6359     \clist_map_inline:Nn \l_@@_cols_tl
6360     {
6361         \@@_qpoint:n { col - ##1 }
6362         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6363         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6364         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6365         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6366         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6367         \clist_map_inline:Nn \l_@@_rows_tl
6368         {
6369             \@@_if_in_corner:nF { #####1 - ##1 }
6370             {
6371                 \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6372                 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6373                 \@@_qpoint:n { row - #####1 }
6374                 \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6375                 \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
6376                 {
6377                     \pgfpathrectanglecorners
6378                     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6379                     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6380                 }
6381             }
6382         }
6383     }
6384 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor`, etc.

```

6385 \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

```

6386 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6387 {
6388     \bool_set_true:N \l_@@_nocolor_used_bool

```

```

6389 \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6390 \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6391 \clist_map_inline:Nn \l_@@_rows_tl
6392 {
6393   \clist_map_inline:Nn \l_@@_cols_tl
6394   { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
6395 }
6396 }

```

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.

```

6397 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6398 {
6399   \clist_set_eq:NN \l_tmpa_clist #1
6400   \clist_clear:N #1
6401   \clist_map_inline:Nn \l_tmpa_clist
6402   {

```

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

```

6403   \def \l_tmpa_tl { ##1 }
6404   \tl_if_in:NnTF \l_tmpa_tl { - }
6405   { \@@_cut_on_hyphen:w ##1 \q_stop }
6406   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6407   \bool_lazy_or:nnT
6408   { \str_if_eq_p:ee \l_tmpa_tl { * } }
6409   { \tl_if_blank_p:o \l_tmpa_tl }
6410   { \def \l_tmpa_tl { 1 } }
6411   \bool_lazy_or:nnT
6412   { \str_if_eq_p:ee \l_tmpb_tl { * } }
6413   { \tl_if_blank_p:o \l_tmpb_tl }
6414   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6415   \int_compare:nNnT \l_tmpb_tl > { #2 }
6416   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6417   \int_step_tokens:nnn \l_tmpa_tl \l_tmpb_tl { \clist_put_right:Nn #1 }
6418 }
6419 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6420 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6421 {
6422   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6423   {

```

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`).

```

6424   \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6425   { \int_use:N \c@iRow - \int_use:N \c@jCol }
6426 }
6427 \ignorespaces
6428 }

6429 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6430 { \@@_error:n { cellcolor~in~Block } }
6431 % \end{macrocode}
6432 %
6433 % \begin{macrocode}
6434 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6435 { \@@_error:n { rowcolor~in~Block } }
6436 % \end{macrocode}
6437 %
6438 % \bigskip

```

```

6439 % The following command will be linked to |\rowcolor| in the tabular.
6440 % \begin{macrocode}
6441 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6442 {
6443   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6444   {
6445     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6446     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6447     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6448   }
6449   \ignorespaces
6450 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6451 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6452 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6453 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6454 {

```

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`.

```

6455   \seq_gclear:N \g_tmpa_seq
6456   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6457   { \@@_rowlistcolors_tabular:nnnn #1 }
6458   \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`.

```

6459   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6460   {
6461     { \int_use:N \c@iRow }
6462     { \exp_not:n { #1 } }
6463     { \exp_not:n { #2 } }
6464     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6465   }
6466   \ignorespaces
6467 }

```

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`).

```

6468 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6469 {
6470   \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).

```

6471     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6472     {
6473       \tl_gput_right:Ne \g_@@_pre_code_before_tl
6474       {
6475         \@@_rowlistcolors
6476         [ \exp_not:n { #2 } ]
6477         { #1 - \int_eval:n { \c@iRow - 1 } }
6478         { \exp_not:n { #3 } }
6479         [ \exp_not:n { #4 } ]
6480       }
6481     }
6482 }

```

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.

```

6483 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6484 {
6485   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6486   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6487   \seq_gclear:N \g_@@_rowlistcolors_seq
6488 }
6489 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6490 {
6491   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6492   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6493 }

```

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.

```

6494 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6495 {

```

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").

```

6496   \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6497   {
6498     \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).

```

6499     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6500     {
6501       \exp_not:N \columncolor [ #1 ]
6502       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6503     }
6504   }
6505 }

```

```

6506 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6507 {
6508   \clist_map_inline:nn { #1 }
6509   {

```

```

6510     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6511     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99!
6512     \columncolor { nocolor } { ##1 }
6513   }
6514 }

6515 \cs_new_protected:Npn \@@_EmptyRow:n #1
6516 {
6517   \clist_map_inline:nn { #1 }
6518   {
6519     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6520     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99!
6521     \rowcolor { nocolor } { ##1 }
6522   }
6523 }

```

The following command must *not* be protected.

```

6524 \cs_new:Npn \@@_FalseRow
6525 {
6526   \noalign
6527   {
6528     \skip_vertical:n
6529     { - \box_ht_plus_dp:N \@arstrutbox + \l_@@_width_of_false_dim }
6530   }
6531   \multicolumn { 1 } { @{} c @{} } { } % added 2026-08-21

```

We keep in memory a list of the `\FalseRow` because we use it with `create-blocks-in-col`.

```

6532   \clist_gput_right:Ne \g_@@_false_rows_clist { \arabic { iRow } }
6533   \\\
6534 }

6535 \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`.

```

6536 \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`.

```

6537 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6538 {
6539   \int_if_zero:nTF \l_@@_first_col_int
6540   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6541   {
6542     \int_if_zero:nTF \c@jCol
6543     {
6544       \int_compare:nNnF \c@iRow = { -1 }
6545       {
6546         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6547         { #1 }

```

```

6548     }
6549   }
6550   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6551 }
6552 }

```

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.

```

6553 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6554 {
6555   \int_if_zero:nF \c@iRow
6556   {
6557     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6558     {
6559       \int_compare:nNnT \c@jCol > \c_zero_int
6560       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6561     }
6562   }
6563 }

```

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`.

```

6564 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6565 {
6566   \IfPackageLoadedTF { tikz }
6567   {
6568     \IfPackageLoadedTF { booktabs }
6569     { #2 }
6570     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6571   }
6572   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6573 }
6574 \NewExpandableDocumentCommand { \@@_TopRule } { }
6575 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6576 \cs_new:Npn \@@_TopRule_i:
6577 {
6578   \noalign \bgroup
6579   \peek_meaning:NTF [
6580   { \@@_TopRule_ii: }
6581   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6582 }
6583 \NewDocumentCommand \@@_TopRule_ii: { o }
6584 {
6585   \tl_gput_right:Ne \g_@@_rules_tl
6586   {
6587     \@@_draw_hrulerule:n
6588     {
6589       position = \int_eval:n { \c@iRow + 1 } ,
6590       tikz =
6591       {
6592         line-width = #1 ,
6593         yshift = 0.25 \arrayrulewidth ,
6594         shorten-< = - 0.5 \arrayrulewidth
6595       } ,
6596       total-width = #1
6597     }
6598   }
6599   \skip_vertical:n { \belowrulesep + #1 }

```

```

6600 \egroup
6601 }
6602 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6603 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6604 \cs_new:Npn \@@_BottomRule_i:
6605 {
6606   \noalign \bgroup
6607   \peek_meaning:NTF [
6608     { \@@_BottomRule_ii: }
6609     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6610   }
6611 \NewDocumentCommand \@@_BottomRule_ii: { o }
6612 {
6613   \tl_gput_right:Ne \g_@@_rules_tl
6614   {
6615     \@@_draw_hrulerule:n
6616     {
6617       position = \int_eval:n { \c@iRow + 1 } ,
6618       tikz =
6619       {
6620         line-width = #1 ,
6621         yshift = 0.25 \arrayrulewidth ,
6622         shorten-< = - 0.5 \arrayrulewidth
6623       } ,
6624       total-width = #1 ,
6625     }
6626   }
6627   \skip_vertical:N \aboverulesep
6628   \@@_create_row_node_i:
6629   \skip_vertical:n { #1 }
6630   \egroup
6631 }
6632 \NewExpandableDocumentCommand { \@@_MidRule } { }
6633 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6634 \cs_new:Npn \@@_MidRule_i:
6635 {
6636   \noalign \bgroup
6637   \peek_meaning:NTF [
6638     { \@@_MidRule_ii: }
6639     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6640   }
6641 \NewDocumentCommand \@@_MidRule_ii: { o }
6642 {
6643   \skip_vertical:N \aboverulesep
6644   \@@_create_row_node_i:
6645   \tl_gput_right:Ne \g_@@_rules_tl
6646   {
6647     \@@_draw_hrulerule:n
6648     {
6649       position = \int_eval:n { \c@iRow + 1 } ,
6650       tikz =
6651       {
6652         line-width = #1 ,
6653         yshift = 0.25 \arrayrulewidth ,
6654         shorten-< = - 0.5 \arrayrulewidth
6655       } ,
6656       total-width = #1 ,
6657     }
6658   }
6659   \skip_vertical:n { \belowrulesep + #1 }
6660   \egroup
6661 }

```

General system for drawing rules

```

6662 \AtBeginDocument
6663 {
6664   \cs_new_protected:Npe \@@_draw_rules:
6665   {
6666     \c_@@_pgfortikzpicture_tl
6667     \@@_draw_rules_i:
6668     \c_@@_endpgfortikzpicture_tl
6669   }
6670 }

6671 \cs_new_protected:Npn \@@_draw_rules_i:
6672 {
6673   \bool_lazy_all:nT
6674   {
6675     { \clist_if_empty_p:N \l_@@_corners_clist }
6676     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6677     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6678     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6679     { ! \l_@@_fix_vertex_bool }
6680   }
6681   {
6682     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6683     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6684   }
6685   \pgfrememberpicturepositiononpagetrue
6686   \pgf@relevantforpicturesizefalse
6687   \pgfsetrectcap
6688   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6689   \CT@arc@
6690   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6691   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6692   \g_@@_rules_tl
6693 }

```

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”.

```

6694 \keys_define:nn { nicematrix / rules-after }
6695 {
6696   position .int_set:N = \l_@@_position_int ,
6697   start .int_set:N = \l_@@_start_int ,
6698   start .initial:n = 1 ,
6699   end .int_set:N = \l_@@_end_int ,
6700   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6701   dotted .code:n =
6702     \bool_set_true:N \l_@@_dotted_bool
6703     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6704     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6705   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]`.

```

6706   color .code:n =
6707     \@@_set_CTarc:n { #1 }
6708     \CT@arc@
6709     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6710   color .value_required:n = true ,
6711   sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6712   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
  }
}

6713 total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6714 unknown .code:n =
6715   \@@_unknown_key:nn
6716     { nicematrix / rules-after }
6717     { Unknown-key-for~a~rule } ,
6718 tikz .value_required:n = true
6719 }
```

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.

```
6720 \AtBeginDocument
6721 {
6722   \IfPackageLoadedTF { tikz }
6723   {
6724     \keys_define:nn { nicematrix / rules-after }
6725     {
6726       tikz .code:n =
6727         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6728         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6729         \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6730 dashed .meta:n =
6731       {
6732         tikz = nicematrix / dashed ,
6733         total-width = \pgflinewidth
6734       }
6735     }
6736   }
6737   {
6738     \keys_define:nn { nicematrix / rules-after }
6739     { tikz .code:n = \@@_error:n { tikz-without~tikz } }
6740   }
6741 }

6742 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6743 {
6744   \int_set:Nn \l_@@_multiplicity_int { #1 }
6745   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6746   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6747 }

6748 \AddToHook { package / tikz / after }
6749 {
6750   \tikzset
6751   {
6752     nicematrix / dashed / .style =
6753     {
6754       dash-pattern = on~4~pt~off~2pt ,
6755       line-cap = butt
6756     }
6757   }
6758   \cs_if_exist:cT { tikz@library@decorations@loaded }
6759   { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6760 }
```

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.

```
6761 \cs_new_protected:Npn \@@_draw_vrule:n #1
6762 {
```

The group is for the options.

```
6763 {
6764   \int_set_eq:NN \l_@@_end_int \c_iRow
6765   \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).

```
6766   \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
6767     { \socket_use:n { nicematrix / draw-vrule } }
6768   }
6769 }
```

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.

```
6770 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6771 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6772 {
6773   \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`.

```
6774   \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).

```
6775   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6776   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6777   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6778     \l_tmpa_tl
6779   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```
6780   \@@_test_v:
6781   \bool_if:NTF \g_tmpa_bool
6782     {
6783     \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.

```
6784     { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
```

```

6785         { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6786     }
6787     {
6788         \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6789         {
6790             \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6791         \socket_use:n { nicematrix / vsegment }
6792         \int_zero:N \l_@@_segment_start_int
6793     }
6794 }
6795 }
6796 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6797 {
6798     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6799     \socket_use:n { nicematrix / vsegment }
6800 }
6801 \group_end:
6802 }
6803 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6804 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6805 {
6806     \group_begin:
6807     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6808     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6809     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6810     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6811     \socket_use:n { nicematrix / vsegment }
6812     \group_end:
6813 }

```

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.

```

6814 \cs_new_protected:Npn \@@_test_v:
6815 {
6816     \bool_gset_true:N \g_tmpa_bool
6817     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6818     \bool_if:NT \g_tmpa_bool
6819     {
6820         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6821         { \@@_test_vline_in_block:nnnnn ##1 }
6822         \bool_if:NT \g_tmpa_bool
6823         {
6824             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6825             { \@@_test_vline_in_block:nnnnn ##1 }
6826             \bool_if:NT \g_tmpa_bool
6827             {
6828                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6829                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6830             }
6831         }
6832     }
6833 }
6834 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6835 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6836 {
6837     {
6838         \int_compare:nNnTF \l_@@_position_int > \c@jCol
6839         { \bool_set_false:N \l_tmpa_bool }

```

```

6840     {
6841       \@@_test_h:
6842       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6843     }
6844     \int_compare:nNnTF \l_@@_position_int = 1
6845     { \bool_gset_false:N \g_tmpa_bool }
6846     {
6847       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6848       \@@_test_h:
6849     }
6850     \bool_gset:Nn \g_tmpa_bool
6851     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6852   }
6853   \bool_if:NT \g_tmpa_bool
6854   {
6855     \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6856     \socket_use:n { nicematrix / vsegment }
6857     \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6858   }
6859 }

6860 \cs_new_protected:Npn \@@_test_in_corner_v:
6861 {
6862   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6863   {
6864     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6865     { \bool_set_false:N \g_tmpa_bool }
6866   }
6867   {
6868     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6869     {
6870       \int_compare:nNnTF \l_tmpb_tl = 1
6871       { \bool_set_false:N \g_tmpa_bool }
6872       {
6873         \@@_if_in_corner:nT
6874         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6875         { \bool_set_false:N \g_tmpa_bool }
6876       }
6877     }
6878   }
6879 }

```

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`.

```

6880 \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`.

```

6881 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6882 {
6883   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6884   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6885   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6886   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6887   \pgfusepathqstroke
6888 }

```

```

6889 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6890 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6891 {
6892   \dim_add:Nn \l_@@_x_initial_dim
6893   {
6894     - 0.5 \l_@@_rule_width_after_dim
6895     +
6896     ( \arrayrulewidth * \l_@@_multiplicity_int
6897       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6898   }
6899   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6900   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6901   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6902   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6903   \bool_lazy_and:nnT
6904   { \cs_if_exist_p:N \CT@drsc@ }
6905   { ! \tl_if_blank_p:o \CT@drsc@ }
6906   {
6907     {
6908       \CT@drsc@
6909       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6910       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6911       \dim_set:Nn \l_@@_tmpd_dim
6912       {
6913         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6914         * ( \l_@@_multiplicity_int - 1 )
6915       }
6916       \pgfpathrectanglecorners
6917       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6918       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6919       \pgfusepath { fill }
6920     }
6921   }
6922   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6923   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6924   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6925   {
6926     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6927     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6928     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6929   }
6930   \pgfusepathqstroke
6931 }

6932 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6933 {
6934   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6935   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6936   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6937   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6938   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6939   \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`

```
6940 \@@_draw_line:
6941 }
```

```
6942 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6943 {
6944 }
```

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).

```
6945 \tl_if_empty:NF \l_@@_rule_color_tl
6946 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6947 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6948 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6949 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6950 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
```

The following line can't be short-cutted.

```
6951 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6952 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6953 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6954 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6955 }
6956 }
```

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).

```
6957 \cs_new_protected:Npn \@@_draw_key_vlines:
6958 {
6959 {
6960 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6961 \int_set:Nn \l_@@_end_int \c@iRow
```

There is a currying in the following code.

```
6962 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6963 { \@@_lines_step_inline:n \c@jCol }
6964 { \clist_map_inline:Nn \l_@@_vlines_clist }
6965 {
6966 \int_set:Nn \l_@@_position_int { ##1 }
6967 \socket_use:n { nicematrix / draw-vrule }
6968 }
6969 }
6970 }
```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```
6971 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6972 {
6973 \int_step_inline:nnn
6974 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6975 {
6976 #1
6977 \bool_lazy_or:nnF \g_@@_delims_bool \l_@@_except_borders_bool
6978 { + 1 }
6979 }
6980 }
```

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.

```

6981 \cs_new_protected:Npn \@@_draw_hrule:n #1
6982 {
6983   {
6984     \int_set_eq:NN \l_@@_end_int \c@jCol
6985     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6986     \socket_use:nn { nicematrix / draw-hrule }
6987   }
6988 }
```

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.

```

6989 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6990 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6991 {
6992   \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`.

```

6993   \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).

```

6994   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6995   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6996   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6997     \l_tmpb_tl
6998   {
```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6999     \@@_test_h:
7000     \bool_if:NTF \g_tmpa_bool
7001     {
7002       \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.

```

7003       { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
7004       { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
7005     }
7006   {
7007     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
7008     {
7009       \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
```

The socket `hsegment` is defined below with its four plugs.

```

7010         \socket_use:n { nicematrix / hsegment }
7011         \int_zero:N \l_@@_segment_start_int
7012     }
7013 }
7014 }
7015 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
7016 {
7017     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7018     \socket_use:n { nicematrix / hsegment }
7019 }
7020 \group_end:
7021 }
7022 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
7023 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
7024 {
7025     \group_begin:
7026     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
7027     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7028     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
7029     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
7030     \socket_use:n { nicematrix / hsegment }
7031     \group_end:
7032 }

```

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.

```

7033 \cs_new_protected:Npn \@@_test_h:
7034 {
7035     \bool_gset_true:N \g_tmpa_bool
7036     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
7037     \bool_if:NT \g_tmpa_bool
7038     {
7039         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7040         { \@@_test_hline_in_block:nnnnn ##1 }
7041         \bool_if:NT \g_tmpa_bool
7042         {
7043             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
7044             { \@@_test_hline_in_block:nnnnn ##1 }
7045             \bool_if:NT \g_tmpa_bool
7046             {
7047                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
7048                 { \@@_test_hline_in_stroken_block:nnnn ##1 }
7049             }
7050         }
7051     }
7052 }
7053 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
7054 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
7055 {
7056     {
7057         \int_compare:nNnTF \l_@@_position_int > \c@iRow
7058         { \bool_set_false:N \l_tmpa_bool }
7059         {
7060             \@@_test_v:
7061             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
7062         }
7063         \int_compare:nNnTF \l_@@_position_int = 1

```

```

7064         { \bool_gset_false:N \g_tmpa_bool }
7065         {
7066             \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
7067             \@@_test_v:
7068         }
7069         \bool_gset:Nn \g_tmpa_bool
7070         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
7071     }
7072     \bool_if:NT \g_tmpa_bool
7073     {
7074         \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
7075         \socket_use:n { nicematrix / hsegment }
7076         \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
7077     }
7078 }

7079 \cs_new_protected:Npn \@@_test_in_corner_h:
7080 {
7081     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
7082     {
7083         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7084         { \bool_set_false:N \g_tmpa_bool }
7085     }
7086     {
7087         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
7088         {
7089             \int_compare:nNnTF \l_tmpa_tl = 1
7090             { \bool_set_false:N \g_tmpa_bool }
7091             {
7092                 \@@_if_in_corner:nT
7093                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
7094                 { \bool_set_false:N \g_tmpa_bool }
7095             }
7096         }
7097     }
7098 }

```

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` or `custom-line`.

```

7099 \socket_new:nn { nicematrix / hsegment } { 0 }

```

In the following plug, the y -value for the line has been computed previously in `\l_@@_y_initial_dim`.

```

7100 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
7101 {
7102     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7103     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7104     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7105     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
7106     \pgfusepathqstroke
7107 }
7108 \socket_assign_plug:nn { nicematrix / hsegment } { standard }

```

```

7109 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
7110 {
7111   \dim_add:Nn \l_@@_y_initial_dim
7112   {
7113     - 0.5 \l_@@_rule_width_after_dim
7114     +
7115     ( \arrayrulewidth * \l_@@_multiplicity_int
7116       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
7117   }
7118   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7119   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7120   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7121   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7122   \bool_lazy_and:nnT
7123   { \cs_if_exist_p:N \CT@drsc@ }
7124   { ! \tl_if_blank_p:o \CT@drsc@ }
7125   {
7126     {
7127       \CT@drsc@
7128       \dim_set:Nn \l_@@_tmpd_dim
7129       {
7130         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
7131         * ( \l_@@_multiplicity_int - 1 )
7132       }
7133       \pgfpathrectanglecorners
7134       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7135       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
7136       \pgfusepathqfill
7137     }
7138   }
7139   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7140   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7141   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
7142   {
7143     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
7144     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
7145     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
7146   }
7147   \pgfusepathqstroke
7148 }

```

Now, the case of a dotted line.

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
```

```
1 & 2 & 3 & 4 \\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

```

7149 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
7150 {
7151   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
7152   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
7153   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }

```

```

7154 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7155 \int_compare:nNnT \l_@@_segment_start_int = 1
7156 {
7157   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
7158   \bool_if:NF \g_@@_delims_bool
7159   { \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.

```

7160   \tl_if_eq:NnF \g_@@_left_delim_tl (
7161     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
7162   )
7163   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7164   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7165   \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
7166   {
7167     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
7168     \bool_if:NF \g_@@_delims_bool
7169     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
7170     \tl_if_eq:NnF \g_@@_right_delim_tl )
7171     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
7172   }

```

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`

```

7173 \@@_draw_line:
7174 }

```

```

7175 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
7176 {
7177   {

```

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).

```

7178   \tl_if_empty:NF \l_@@_rule_color_tl
7179   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
7180   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
7181   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7182   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
7183   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
7184   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7185   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
7186   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
7187   -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
7188 }
7189 }

```

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).

```

7190 \cs_new_protected:Npn \@@_draw_key_hlines:

```

```

7191 {
7192 {
7193 \dim_set_eq:Nn \l_@@_rule_width_after_dim \arrayrulewidth
7194 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

7195 \str_if_eq:eeTF \l_@@_hlines_clist { all }
7196 { \@@_lines_step_inline:n \c@iRow }
7197 { \clist_map_inline:Nn \l_@@_hlines_clist }
7198 {
7199 \int_set:Nn \l_@@_position_int { ##1 }
7200 \socket_use:n { nicematrix / draw-hrule }
7201 }
7202 }
7203 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

7204 \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.

```

7205 \cs_set:Npn \@@_Hline_i:n #1
7206 {
7207 \peek_remove_spaces:n
7208 {
7209 \peek_meaning:NTF \Hline
7210 { \@@_Hline_ii:nn { #1 + 1 } }
7211 { \@@_Hline_iii:n { #1 } }
7212 }
7213 }
7214 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
7215 \cs_set:Npn \@@_Hline_iii:n #1
7216 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
7217 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
7218 {
7219 \@@_compute_rule_width:n { multiplicity = #1 , #2 }
7220 \skip_vertical:N \l_@@_rule_width_before_dim
7221 \tl_gput_right:Nx \g_@@_rules_tl
7222 {

```

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:

```

7223 {
7224 \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
7225 \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
7226 \dim_set:Nn \l_@@_rule_width_after_dim
7227 { \dim_use:N \l_@@_rule_width_before_dim }
7228 \@@_draw_hrule:n { #2 }
7229 }
7230 }
7231 \egroup
7232 }

```

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`).

```

7233 \cs_new_protected:Npn \@@_custom_line:n #1
7234 {
7235   \str_clear:N \l_@@_letter_str
7236   \str_clear:N \l_@@_command_str
7237   \str_clear:N \l_@@_ccommand_str
7238   \tl_clear_new:N \l_@@_other_keys_tl
7239   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
7240   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
7241   {
7242     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

7243     \str_set:Ne \l_@@_command_str
7244     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
7245   }
7246   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
7247   {
7248     \str_set:Ne \l_@@_ccommand_str
7249     { \str_tail:N \l_@@_ccommand_str }
7250     \str_set:Ne \l_@@_ccommand_str
7251     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
7252   }

```

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.

```

7253   \bool_lazy_all:nTF
7254   {
7255     { \str_if_empty_p:N \l_@@_letter_str }
7256     { \str_if_empty_p:N \l_@@_command_str }
7257     { \str_if_empty_p:N \l_@@_ccommand_str }
7258   }
7259   { \@@_error:n { No~letter~and~no~command } }
7260   { \@@_custom_line_i:o \l_@@_other_keys_tl }
7261 }

7262 \keys_define:nn { nicematrix / custom-line }
7263 {
7264   letter .str_set:N = \l_@@_letter_str ,
7265   letter .value_required:n = true ,
7266   command .str_set:N = \l_@@_command_str ,
7267   command .value_required:n = true ,
7268   ccommand .str_set:N = \l_@@_ccommand_str ,
7269   ccommand .value_required:n = true
7270 }

```

```

7271 \cs_new_protected:Npn \@@_custom_line_i:n #1
7272 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

7273   \bool_set_false:N \l_@@_tikz_rule_bool
7274   \bool_set_false:N \l_@@_dotted_rule_bool
7275   \bool_set_false:N \l_@@_color_bool

```

```

7276 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
7277 \bool_if:NT \l_@@_tikz_rule_bool
7278 {
7279   \IfPackageLoadedF { tikz }
7280   { \@@_error:n { tikz-in-custom-line-without-tikz } }
7281   \bool_if:NT \l_@@_color_bool
7282   { \@@_error:n { color-in-custom-line-with-tikz } }
7283 }
7284 \bool_if:NT \l_@@_dotted_rule_bool
7285 {
7286   \int_compare:nNnT \l_@@_multiplicity_int > 1
7287   { \@@_error:n { key-multiplicity-with-dotted } }
7288 }
7289 \str_if_empty:NF \l_@@_letter_str
7290 {
7291   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
7292   { \@@_error:n { Several-letters } }
7293   {
7294     \tl_if_in:NoTF
7295     \c_@@_forbidden_letters_str
7296     \l_@@_letter_str
7297     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
7298   }

```

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.

```

7299         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7300         { \@@_v_custom_line:nn { #1 } }
7301     }
7302 }
7303 }
7304 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7305 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7306 }
7307 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7308 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbFVX|()[]!@<> }
7309 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbFVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/rules-after}`). That's why the following set of keys has some keys which are no-op.

```

7310 \keys_define:nn { nicematrix / custom-line-bis }
7311 {
7312   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7313   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7314   color .value_required:n = true ,
7315   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7316   tikz .value_required:n = true ,
7317   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7318   dotted .value_forbidden:n = true ,
7319   dashed .meta:n =
7320   {
7321     tikz = nicematrix / dashed ,
7322     total-width = \pgflinewidth
7323   } ,
7324   total-width .code:n = { } ,
7325   total-width .value_required:n = true ,
7326   sep-color .code:n = { } ,
7327   sep-color .value_required:n = true ,
7328   unknown .code:n =
7329   \@@_unknown_key:nn

```

```

7330      { nicematrix / custom-line-bis }
7331      { Unknown-key-for-custom-line }
7332  }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7333 \bool_new:N \l_@@_dotted_rule_bool
7334 \bool_new:N \l_@@_tikz_rule_bool
7335 \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).

```

7336 \keys_define:nn { nicematrix / custom-line-width }
7337 {
7338   multiplicity .int_set:N = \l_@@_tmpc_int ,
7339   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7340   total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7341                       \bool_set_true:N \l_@@_total_width_bool ,
7342   total-width .value_required:n = true ,
7343   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7344 }

```

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_hrrule:n` (which is in the internal `\CodeAfter`).

```

7345 \cs_new_protected:Npn \@@_h_custom_line:n #1
7346 {

```

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`).

```

7347   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7348   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7349 }

```

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_hrrule:n` (which is in the internal `\CodeAfter`).

```

7350 \cs_new_protected:Npn \@@_c_custom_line:n #1
7351 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7352   \exp_args:Nc \DeclareExpandableDocumentCommand
7353   { nicematrix - \l_@@_ccommand_str }
7354   { 0 { } m }
7355   {
7356     \noalign
7357     {
7358       \@@_compute_rule_width:n { #1 , ##1 }
7359       \skip_vertical:n \l_@@_rule_width_before_dim
7360       \clist_map_inline:nn
7361       { ##2 }
7362       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7363     }
7364   }
7365   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7366 }

```

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*.

```

7367 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7368 {
7369   \tl_if_in:nnTF { #2 } { - }
7370   { \@@_cut_on_hyphen:w #2 \q_stop }
7371   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7372   \tl_gput_right:Ne \g_@@_rules_tl
7373   {
7374     \@@_draw_hrule:n
7375     {
7376       #1 ,
7377       start = \l_tmpa_tl ,
7378       end = \l_tmpb_tl ,
7379       position = \int_eval:n { \c@iRow + 1 } ,
7380       total-width = \dim_use:N \l_@@_rule_width_before_dim
7381     }
7382   }
7383 }

7384 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7385 {
7386   \bool_set_false:N \l_@@_tikz_rule_bool
7387   \bool_set_false:N \l_@@_total_width_bool
7388   \bool_set_false:N \l_@@_dotted_rule_bool
7389   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7390   \bool_if:NF \l_@@_total_width_bool
7391   {
7392     \bool_if:NTF \l_@@_dotted_rule_bool
7393     { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7394     {
7395       \bool_if:NF \l_@@_tikz_rule_bool
7396       {
7397         \dim_set:Nn \l_@@_rule_width_before_dim
7398         {
7399           \arrayrulewidth * \l_@@_tmpc_int
7400           + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7401         }
7402       }
7403     }
7404   }
7405 }

```

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

```

7406 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7407 {
7408   \str_if_eq:nnTF { #2 } { [ ] }
7409   { \@@_v_custom_line_i:nw { #1 } [ ] }
7410   { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7411 }
7412 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7413 { \@@_v_custom_line:nn { #1 , #2 } }
7414 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7415 {
7416   \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7417 \tl_put_right:Ne \l_@@_array_preamble_tl
7418 {
7419   \exp_not:N !
7420   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7421 }
7422 \tl_gput_right:Ne \g_@@_rules_tl

```

```
7423 {
```

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.

```
7424 {
7425   \dim_set:Nn \l_@@_rule_width_after_dim
7426   { \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).

```
7427   \int_set:Nn \l_@@_position_int
7428   { \int_eval:n { \g_@@_static_num_of_col_int + 1 } }
7429   \@@_draw_vrule:n { #2 }
7430 }
7431 }
7432 \@@_rec_preamble:n #1
7433 }

7434 \@@_custom_line:n
7435 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }
```

The key `default-line`

```
7436 \keys_define:nn { nicematrix / default-line }
7437 {
7438   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7439   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7440   color .value_required:n = true ,
7441   dashed .code:n = \@@_error:n { dashed-for-default-line-without-TikZ } ,
7442   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7443   dotted .value_forbidden:n = true ,
7444   total-width .code:n = { } ,
7445   total-width .value_required:n = true ,
7446   sep-color .code:n = { } ,
7447   sep-color .value_required:n = true ,
7448   unknown .code:n =
7449     \@@_unknown_key:nn
7450     { nicematrix / default-line }
7451     { Unknown-key-for-default-line }
7452 }

7453 \AddToHook { package / tikz / after }
7454 {
7455   \keys_define:nn { nicematrix / default-line }
7456   {
7457     tikz .code:n =
7458       \bool_set_true:N \l_@@_tikz_rule_bool
7459       \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7460     tikz .value_required:n = true ,
7461     dashed .meta:n =
7462       {
7463         tikz = nicematrix / dashed ,
7464         total-width = \pgflinewidth
7465       }
7466   }
7467   \@@_custom_line:n
7468   {
```

```

7469         letter = ; ,
7470         command = \hdashedline ,
7471         ccommand = \cdashedline ,
7472         tikz = nicematrix / dashed ,
7473         total-width = \pgflinewidth
7474     }
7475 }

7476 \cs_new_protected:Npn \@@_default_line:n #1
7477 {
7478     \bool_set_false:N \l_@@_tikz_rule_bool
7479     \bool_set_false:N \l_@@_dotted_rule_bool
7480     \bool_set_false:N \l_@@_color_bool
7481     \keys_set:nn { nicematrix / default-line } { #1 }
7482     \bool_if:NT \l_@@_tikz_rule_bool
7483     {
7484         \IfPackageLoadedF { tikz }
7485         { \@@_error:n { tikz~in~custom~line~without~tikz } }
7486         \bool_if:NT \l_@@_color_bool
7487         { \@@_error:n { color~in~custom~line~with~tikz } }
7488     }
7489     \bool_if:NT \l_@@_dotted_rule_bool
7490     {
7491         \int_compare:nNnT \l_@@_multiplicity_int > 1
7492         { \@@_error:n { key~multiplicity~with~dotted } }
7493     }
7494     \bool_if:NTF \l_@@_tikz_rule_bool
7495     {
7496         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7497         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7498     }
7499     {
7500         \bool_if:NTF \l_@@_dotted_rule_bool
7501         {
7502             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7503             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7504         }
7505         {
7506             \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7507             \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7508         }
7509     }
7510 }

```

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.

```

7511 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7512 {
7513   \if_int_compare:w \l_tmpa_tl > #1
7514     \if_int_compare:w \l_tmpa_tl > #3 \else:
7515       \if_int_compare:w \l_tmpb_tl < #2 \else:
7516         \if_int_compare:w \l_tmpb_tl > #4 \else:
7517           \bool_gset_false:N \g_tmpa_bool
7518         \fi:
7519       \fi:
7520     \fi:
7521   \fi:
7522 }

```

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.

```

7523 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7524 {
7525   \if_int_compare:w \l_tmpa_tl < #1 \else:
7526     \if_int_compare:w \l_tmpa_tl > #3 \else:
7527       \if_int_compare:w \l_tmpb_tl > #2
7528         \if_int_compare:w \l_tmpb_tl > #4 \else:
7529           \bool_gset_false:N \g_tmpa_bool
7530         \fi:
7531       \fi:
7532     \fi:
7533   \fi:
7534 }

```

Now, for the stroken blocks.

```

7535 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7536 {
7537   % \int_compare:nNnT \l_tmpb_tl < { #2 - 1 }
7538   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7539   {
7540     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7541     {
7542       \int_compare:nNnTF \l_tmpa_tl = { #1 }
7543       { \bool_gset_false:N \g_tmpa_bool }
7544       {
7545         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }

```

```

7546         { \bool_gset_false:N \g_tmpa_bool }
7547     }
7548 }
7549 }
7550 }
7551 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7552 {
7553     \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
7554     {
7555         \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
7556         {
7557             \int_compare:nNtTF \l_tmpb_tl = { #2 }
7558             { \bool_gset_false:N \g_tmpa_bool }
7559             {
7560                 \int_compare:nNtT \l_tmpb_tl = { #4 + 1 }
7561                 { \bool_gset_false:N \g_tmpa_bool }
7562             }
7563         }
7564     }
7565 }

```

23 The empty corners

```

7566 \cs_new_protected:Npn \@@_mark_blocks:
7567 {
7568     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7569     { \@@_mark_cells_of_block:nnnnn ##1 }
7570
7571     \cs_set_eq:NN \@@_mark_blocks: \prg_do_nothing:
7572 }

```

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.

```

7573 \cs_new_protected:Npn \@@_compute_corners:
7574 {
7575     \@@_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.

```

7576     \clist_clear:N \l_@@_corners_cells_clist
7577     \clist_map_inline:Nn \l_@@_corners_clist
7578     {
7579         \str_case:nnF { ##1 }
7580         {
7581             { NW }
7582             { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7583             { NE }
7584             { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7585             { SW }
7586             { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7587             { SE }
7588             { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7589             { N }
7590             { \@@_compute_corner_NS:nnnn \c@jCol 1 1 \c@iRow }
7591             { S }
7592             { \@@_compute_corner_NS:nnnn \c@jCol \c@iRow { -1 } 1 }
7593             { E }

```

```

7594 { \@@_compute_corner_EW:nnnn \c@iRow \c@jCol { -1 } 1 }
7595 { W }
7596 { \@@_compute_corner_EW:nnnn \c@iRow 1 1 \c@jCol}
7597 }
7598 { \@@_error:nn { bad~corner } { ##1 } }
7599 }

```

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`.

```

7600 \clist_if_empty:NF \l_@@_corners_cells_clist
7601 {

```

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.

```

7602 \tl_gput_right:Ne \g_@@_aux_tl
7603 {
7604 \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7605 { \l_@@_corners_cells_clist }
7606 }
7607 }
7608 }

```

```

7609 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7610 {
7611 \int_step_inline:nnn { #1 } { #3 }
7612 {
7613 \int_step_inline:nnn { #2 } { #4 }
7614 { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7615 }
7616 }

```

```

7617 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p , TF , T , F }
7618 {
7619 \cs_if_exist:cTF
7620 { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7621 \prg_return_true:
7622 \prg_return_false:
7623 }

```

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).

```

7624 \cs_new_protected:Npn \@@_mark_empty_cell:nn #1 #2
7625 {
7626 \clist_put_right:Ne \l_@@_corners_cells_clist { #1 - #2 }
7627 \cs_set_nopar:cpn { @@ _ corner _ #1 - #2 } { }
7628 }

```

“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.

```

7629 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7630 {

```

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.

```

7631 \int_zero_new:N \l_@@_last_empty_row_int
7632 \int_set:Nn \l_tmpa_int { #1 }
7633 \bool_set_false:N \l_tmpa_bool
7634 \bool_do_until:Nn \l_tmpa_bool
7635 {
7636   \bool_lazy_or:nnTF
7637   {
7638     \cs_if_exist_p:c
7639     {
7640       pgf @ sh @ ns @ \@@_env: -
7641       \int_use:N \l_tmpa_int - \int_eval:n { #2 }
7642     }
7643   }
7644   { \@@_if_in_block_p:nn \l_tmpa_int { #2 } }
7645   {
7646     \bool_set_true:N \l_tmpa_bool
7647     \int_compare:nNnTF \l_tmpa_int = { #1 }
7648     { \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int }
7649     { \int_set:Nn \l_@@_last_empty_row_int { \l_tmpa_int - #3 } }
7650   }
7651   {
7652     \int_compare:nNnTF \l_tmpa_int = { #5 }
7653     {
7654       \bool_set_true:N \l_tmpa_bool
7655       \int_set_eq:NN \l_@@_last_empty_row_int \l_tmpa_int
7656     }
7657     { \int_add:Nn \l_tmpa_int { #3 } }
7658   }
7659 }

```

Now, you determine the last empty cell in the row of number 1.

```

7660 \int_zero_new:N \l_@@_last_empty_column_int
7661 \int_set:Nn \l_tmpa_int { #2 }
7662 \bool_set_false:N \l_tmpa_bool
7663 \bool_do_until:Nn \l_tmpa_bool
7664 {
7665   \bool_lazy_or:nnTF
7666   {
7667     \cs_if_exist_p:c
7668     {
7669       pgf @ sh @ ns @ \@@_env: -
7670       \int_eval:n { #1 } - \int_use:N \l_tmpa_int
7671     }
7672   }
7673   { \@@_if_in_block_p:nn { #1 } \l_tmpa_int }
7674   {
7675     \bool_set_true:N \l_tmpa_bool
7676     \int_compare:nNnTF \l_tmpa_int = { #1 }
7677     { \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int }
7678     { \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 } }
7679   }
7680   {
7681     \int_compare:nNnTF \l_tmpa_int = { #6 }
7682     {
7683       \bool_set_true:N \l_tmpa_bool
7684       \int_set_eq:NN \l_@@_last_empty_column_int \l_tmpa_int

```

```

7685         }
7686         { \int_add:Nn \l_tmpa_int { #4 } }
7687     }
7688 }

```

Now, we loop over the rows.

```

7689 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7690 {

```

We treat the row number ##1 with a loop \bool_do_until:Nn.

```

7691     \int_set:Nn \l_tmpa_int { #2 }
7692     \bool_set_false:N \l_tmpa_bool
7693     \bool_do_until:Nn \l_tmpa_bool
7694     {
7695         \bool_lazy_or:nnTF
7696         {
7697             \cs_if_exist_p:c
7698             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7699         }
7700         { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7701         { \bool_set_true:N \l_tmpa_bool }
7702         {
7703             \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7704             \int_compare:nNnTF \l_tmpa_int = \l_@@_last_empty_column_int
7705             { \bool_set_true:N \l_tmpa_bool }
7706             \int_add:Nn \l_tmpa_int { #4 }
7707         }
7708     }
7709     \int_set:Nn \l_@@_last_empty_column_int { \l_tmpa_int - #4 }
7710 }
7711 }

```

For the “corners” N and S

```

7712 \cs_new_protected:Npn \@@_compute_corner_NS:nnnn #1 #2 #3 #4
7713 {
7714     \int_step_inline:nn { #1 }
7715     {
7716         \int_set:Nn \l_tmpa_int { #2 }
7717         \bool_set_false:N \l_tmpa_bool
7718         \bool_do_until:Nn \l_tmpa_bool
7719         {
7720             \bool_lazy_or:nnTF
7721             {
7722                 \cs_if_exist_p:c
7723                 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_tmpa_int - ##1 }
7724             }
7725             { \@@_if_in_block_p:nn \l_tmpa_int { ##1 } }
7726             { \bool_set_true:N \l_tmpa_bool }
7727             {
7728                 \@@_mark_empty_cell:nn { \int_use:N \l_tmpa_int } { ##1 }
7729                 \int_compare:nNnTF \l_tmpa_int = { #4 }
7730                 { \bool_set_true:N \l_tmpa_bool }
7731                 { \int_add:Nn \l_tmpa_int { #3 } }
7732             }
7733         }
7734     }
7735 }

```

For the “corners” E and W.

```

7736 \cs_new_protected:Npn \@@_compute_corner_EW:nnnn #1 #2 #3 #4
7737 {
7738     \int_step_inline:nn { #1 }
7739     {

```

```

7740 \int_set:Nn \l_tmpa_int { #2 }
7741 \bool_set_false:N \l_tmpa_bool
7742 \bool_do_until:Nn \l_tmpa_bool
7743 {
7744   \bool_lazy_or:nnTF
7745   {
7746     \cs_if_exist_p:c
7747     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_tmpa_int }
7748   }
7749   { \@@_if_in_block_p:nn { ##1 } \l_tmpa_int }
7750   { \bool_set_true:N \l_tmpa_bool }
7751   {
7752     \@@_mark_empty_cell:nn { ##1 } { \int_use:N \l_tmpa_int }
7753     \int_compare:nNnTF \l_tmpa_int = { #4 }
7754     { \bool_set_true:N \l_tmpa_bool }
7755     { \int_add:Nn \l_tmpa_int { #3 } }
7756   }
7757 }
7758 }
7759 }

```

Of course, instead of the following lines, we could have used `\prg_new_conditional:Npnn`.

```

7760 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7761 \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.

```
7762 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7763 \keys_define:nn { nicematrix / NiceMatrixBlock }
7764 {
7765   auto-columns-width .code:n =
7766   {
7767     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7768     \dim_gzero_new:N \g_@@_max_cell_width_dim
7769     \bool_set_true:N \l_@@_auto_columns_width_bool
7770   }
7771 }

7772 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7773 {
7774   \int_gincr:N \g_@@_NiceMatrixBlock_int
7775   \dim_zero:N \l_@@_columns_width_dim
7776   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7777   \bool_if:NT \l_@@_block_auto_columns_width_bool
7778   {
7779     \cs_if_exist:cT
7780     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7781     {
7782       \dim_set:Nn \l_@@_columns_width_dim
7783       {

```

```

7784         \use:c
7785         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7786     }
7787 }
7788 }
7789 }

```

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`).

```

7790 {
7791   \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.

```

7792   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7793   {
7794     \bool_if:NT \l_@@_block_auto_columns_width_bool
7795     {
7796       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7797       \iow_shipout:Ne \@mainaux
7798       {
7799         \cs_gset:cpn
7800         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

7801         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7802     }
7803     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7804   }
7805 }
7806 \ignorespacesafterend
7807 }

```

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).

```

7808 \cs_new_protected:Npn \@@_create_extra_nodes:
7809 {
7810   \bool_if:nTF \l_@@_medium_nodes_bool
7811   {
7812     \bool_if:NTF \l_@@_no_cell_nodes_bool
7813     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7814     {
7815       \bool_if:NTF \l_@@_large_nodes_bool
7816       \@@_create_medium_and_large_nodes:
7817       \@@_create_medium_nodes:
7818     }
7819   }
7820   {
7821     \bool_if:NT \l_@@_large_nodes_bool
7822     {
7823       \bool_if:NTF \l_@@_no_cell_nodes_bool
7824       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7825       \@@_create_large_nodes:
7826     }
7827   }
7828 }

```

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`.

```

7829 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7830 {
7831   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7832   {
7833     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7834     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7835     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7836     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7837   }
7838   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7839   {
7840     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7841     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7842     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7843     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7844   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7845   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7846   {
7847     \int_step_variable:nnNn
7848     \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.

```

7849     {
7850       \cs_if_exist:cT
7851       { 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`.

```

7852     {
7853       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7854       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7855       { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7856       \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7857       {
7858         \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7859         { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7860       }

```

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`.

```

7861       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7862       \dim_set:cn { l_@@_row_ \@@_i: _max_dim }
7863       { \dim_max:vn { l_@@_row_ \@@_i: _max_dim } \pgf@y }

```

```

7864         \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7865         {
7866             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7867             { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
7868         }
7869     }
7870 }
7871 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7872 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7873 {
7874     \dim_compare:nNnT
7875     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7876     {
7877         \@@_qpoint:n { row - \@@_i: - base }
7878         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7879         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7880     }
7881 }
7882 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7883 {
7884     \dim_compare:nNnT
7885     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7886     {
7887         \@@_qpoint:n { col - \@@_j: }
7888         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@x
7889         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@x
7890     }
7891 }
7892 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7893 \cs_new_protected:Npn \@@_create_medium_nodes:
7894 {
7895     \pgfpicture
7896     \pgfrememberpicturepositiononpagetrue
7897     \pgf@relevantforpicturesizefalse
7898     \@@_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”.

```

7899     \tl_set:Nn \l_@@_suffix_tl { -medium }
7900     \@@_create_nodes:
7901     \endpgfpicture
7902 }

```

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:`.

```

7903 \cs_new_protected:Npn \@@_create_large_nodes:
7904 {
7905     \pgfpicture
7906     \pgfrememberpicturepositiononpagetrue
7907     \pgf@relevantforpicturesizefalse
7908     \@@_computations_for_medium_nodes:

```

¹⁶If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7909     \@@_computations_for_large_nodes:
7910     \tl_set:Nn \l_@@_suffix_tl { - large }
7911     \@@_create_nodes:
7912     \endpgfpicture
7913 }
7914 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7915 {
7916     \pgfpicture
7917     \pgfrememberpicturepositiononpagetrue
7918     \pgf@relevantforpicturesizefalse
7919     \@@_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”.

```

7920     \tl_set:Nn \l_@@_suffix_tl { - medium }
7921     \@@_create_nodes:
7922     \@@_computations_for_large_nodes:
7923     \tl_set:Nn \l_@@_suffix_tl { - large }
7924     \@@_create_nodes:
7925     \endpgfpicture
7926 }

```

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.

```

7927 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7928 {
7929     \int_set:Nn \l_@@_first_row_int 1
7930     \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`.

```

7931     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7932     {
7933         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7934         {
7935             (
7936                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7937                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7938             )
7939             / 2
7940         }
7941         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7942         { l_@@_row _ \@@_i: _ min _ dim }
7943     }
7944     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7945     {
7946         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7947         {
7948             (
7949                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7950                 \dim_use:c
7951                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7952             )
7953             / 2
7954         }
7955         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7956         { l_@@_column _ \@@_j: _ max _ dim }
7957     }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7958     \dim_sub:cn
7959     { l_@@_column _ 1 _ min _ dim }
7960     \l_@@_left_margin_dim

```

```

7961 \dim_add:cn
7962 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7963 \l_@@_right_margin_dim
7964 }

```

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).

```

7965 \cs_new_protected:Npn \@@_create_nodes:
7966 {
7967   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7968   {
7969     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7970     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7971       \@@_pgf_rect_node:nnnnn
7972       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7973       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7974       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7975       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7976       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7977       \str_if_empty:NF \l_@@_name_str
7978       {
7979         \pgfnodealias
7980         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7981         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7982       }
7983     }
7984   }
7985   \int_step_inline:nn \c@iRow
7986   {
7987     \pgfnodealias
7988     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7989     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7990   }
7991   \int_step_inline:nn \c@jCol
7992   {
7993     \pgfnodealias
7994     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7995     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7996   }
7997   \pgfnodealias
7998   { \@@_env: - last - last \l_@@_suffix_tl }
7999   { \@@_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 .

```

8000   \seq_map_pairwise_function:NNN
8001   \g_@@_multicolumn_cells_seq
8002   \g_@@_multicolumn_sizes_seq
8003   \@@_node_for_multicolumn:nn
8004 }

```

```

8005 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
8006 {
8007   \cs_set_nopar:Npn \@@_i: { #1 }
8008   \cs_set_nopar:Npn \@@_j: { #2 }
8009 }

```

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”).

```

8010 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
8011 {
8012   \@@_extract_coords_values: #1 \q_stop
8013   \@@_pgf_rect_node:nnnnn
8014     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
8015     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
8016     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
8017     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
8018     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
8019   \str_if_empty:NF \l_@@_name_str
8020   {
8021     \pgfnodealias
8022       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
8023       { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
8024   }
8025 }

```

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).

```

8026 \keys_define:nn { nicematrix / BlockFirstPass }
8027 {
8028   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8029     \bool_set_true:N \l_@@_p_block_bool ,
8030   j .value_forbidden:n = true ,
8031   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8032   l .value_forbidden:n = true ,
8033   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8034   r .value_forbidden:n = true ,
8035   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8036   c .value_forbidden:n = true ,
8037   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8038   L .value_forbidden:n = true ,
8039   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8040   R .value_forbidden:n = true ,
8041   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8042   C .value_forbidden:n = true ,
8043   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8044   t .value_forbidden:n = true ,
8045   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8046   T .value_forbidden:n = true ,
8047   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8048   b .value_forbidden:n = true ,
8049   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8050   B .value_forbidden:n = true ,
8051   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8052   m .value_forbidden:n = true ,
8053   v-center .meta:n = m ,
8054   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8055   p .value_forbidden:n = true ,

```

```

8056      respect-arraystretch .code:n =
8057      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8058      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.

```

8059      color .code:n =
8060      \@@_color:n { #1 }
8061      \tl_set_rescan:Nnn
8062      \l_@@_draw_tl
8063      { \char_set_catcode_other:N ! }
8064      { #1 } ,
8065      color .value_required:n = true ,
8066  }

```

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.

```

8067 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

8068 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
8069 {

```

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).

```

8070   \tl_if_blank:nTF { #2 }
8071   { \@@_Block_ii:nnnnn 1 1 }
8072   {
8073     \tl_if_in:nnTF { #2 } { - }
8074     {
8075       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
8076       \@@_Block_i_czech:w \@@_Block_i:w
8077       #2 \q_stop
8078     }
8079     {
8080       \@@_error:nn { Bad~argument~for~Block } { #2 }
8081       \@@_Block_ii:nnnnn 1 1
8082     }
8083   }
8084   { #1 } { #3 } { #4 }
8085   \ignorespaces
8086 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

8087 \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`.

```

8088 {
8089   \char_set_catcode_active:N -
8090   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
8091 }

```

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.

```

8092 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
8093 {

```

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).

```

8094   \int_set:Nn \l_tmpa_int
8095   {
8096     \bool_lazy_or:nnTF
8097     { \tl_if_blank_p:n { #1 } }
8098     { \str_if_eq_p:ee { * } { #1 } }
8099     { 100 }
8100     { #1 }
8101   }
8102   \int_set:Nn \l_tmpb_int
8103   {
8104     \bool_lazy_or:nnTF
8105     { \tl_if_blank_p:n { #2 } }
8106     { \str_if_eq_p:ee { * } { #2 } }
8107     { 100 }
8108     { #2 }
8109   }

```

If the block is mono-column.

```

8110   \int_compare:nNnTF \l_tmpb_int = 1
8111   {
8112     \tl_if_empty:NNTF \l_@@_hpos_cell_tl
8113     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
8114     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
8115   }
8116   { \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.

```

8117   \keys_set:known:n { nicematrix / BlockFirstPass } { #3 }
8118   \tl_set:Ne \l_tmpa_tl
8119   {
8120     { \int_use:N \c@iRow }
8121     { \int_use:N \c@jCol }
8122     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
8123     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
8124   }

```

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).

```

8125   \bool_set_false:N \l_tmpa_bool
8126   \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
8127   { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
8128   \bool_case:nF
8129   {
8130     \l_tmpa_bool { \@@_Block_vii:eennn }
8131     \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the

p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

8132         \l_@@_X_bool                               { \@@_Block_v:eennn }
8133         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
8134         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
8135         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
8136     }
8137     { \@@_Block_v:eennn }
8138     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
8139 }

```

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.

```

8140 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
8141 {
8142     \int_gincr:N \g_@@_block_box_int
8143     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8144     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
8145     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8146     {
8147         \tl_gput_right:Ne \g_@@_rules_tl
8148         {
8149             \@@_draw_diagbox:nnnnnn
8150             { \int_use:N \c@iRow }
8151             { \int_use:N \c@jCol }
8152             { \int_eval:n { \c@iRow + #1 - 1 } }
8153             { \int_eval:n { \c@jCol + #2 - 1 } }
8154             { \g_@@_row_style_tl \exp_not:n { ##1 } }
8155             { \g_@@_row_style_tl \exp_not:n { ##2 } }
8156         }
8157     }
8158     \box_gclear_new:c
8159     { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

8160     \hbox_gset:cn
8161     { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
8162     {

```

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`).

```

8163         \tl_if_empty:NTF \l_@@_color_tl
8164         { \int_compare:nNnT { #2 } = 1 \set@color }
8165         { \@@_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`.

```

8166         \int_compare:nNnT { #1 } = 1
8167         {

```

```

8168         \int_if_zero:nTF \c@iRow
8169         {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\{\scriptstyle\color{blue} \arabic{jCol}\},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

8170         \cs_set_eq:NN \Block \@@_NullBlock:
8171         \l_@@_code_for_first_row_tl
8172     }
8173     {
8174         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8175         {
8176             \cs_set_eq:NN \Block \@@_NullBlock:
8177             \l_@@_code_for_last_row_tl
8178         }
8179     }
8180     \g_@@_row_style_tl
8181 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8182     \@@_reset_arraystretch:
8183     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

8184     #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.).

```

8185     \@@_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}`.

```

8186     \bool_if:NTF \l_@@_tabular_bool
8187     {
8188         \bool_lazy_all:nTF
8189         {
8190             { \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.

```

8191         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
8192         { ! \g_@@_rotate_bool }
8193     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

8194         {
8195             \use:e
8196             {
Curiously, \exp_not:N is still mandatory when tagging=on.
8197                 \exp_not:N \begin { minipage }
8198                 [ \str_lowercase:f \l_@@_vpos_block_str ]
8199                 { \l_@@_col_width_dim }
8200                 \str_case:on \l_@@_hpos_block_str
8201                 { c \centering r \raggedleft l \raggedright }
8202             }
8203             #5
8204         \end { minipage }
8205     }

```

In the other cases, we use a `{tabular}`.

```

8206         {
8207             \use:e
8208             {
Curiously, \exp_not:N is still mandatory when tagging=on.
8209                 \exp_not:N \begin { tabular }
8210                 [ \str_lowercase:f \l_@@_vpos_block_str ]
8211                 { @ { } \l_@@_hpos_block_str @ { } }
8212             }
8213             #5
8214         \end { tabular }
8215     }
8216 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

8217     {
8218         $ % $
8219         \bool_if:NT \l_@@_small_bool
8220         {
8221             \def \arraystretch { 0.47 }
8222             \dim_set:Nn \arraycolsep { 1.45 pt }
8223         }
8224         \use:e
8225         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

8226             \exp_not:N \begin { array }
8227             [ \str_lowercase:f \l_@@_vpos_block_str ]
8228             { @ { } \l_@@_hpos_block_str @ { } }
8229         }
8230         \@@_tuning_key_small:
8231         #5
8232         \end { array }
8233         $ % $
8234     }
8235 }

```

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).

```

8236     \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.

```

8237     \int_compare:nNnT { #2 } = 1
8238     {
8239         \dim_gset:Nn \g_@@_blocks_wd_dim

```

```

8240     {
8241         \dim_max:nn
8242         \g_@@_blocks_wd_dim
8243         {
8244             \box_wd:c
8245             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8246         }
8247     }
8248 }

```

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.

```

8249     \int_compare:nNtT { #1 } = 1
8250     {
8251         \bool_lazy_any:nT
8252         {
8253             { \str_if_empty_p:N \l_@@_vpos_block_str }
8254             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8255             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8256         }
8257         \@@_adjust_blocks_ht_dp:
8258     }
8259     \seq_gput_right:Ne \g_@@_blocks_seq
8260     {
8261         \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.

```

8262     {
8263         \exp_not:n { #3 } ,
8264         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

8265     \bool_if:NT \g_@@_rotate_bool
8266     {
8267         \bool_if:NTF \g_@@_rotate_c_bool
8268         { m }
8269         {
8270             \int_compare:nNtT \c@iRow = \l_@@_last_row_int
8271             { T }
8272         }
8273     }
8274     {
8275         {
8276             \box_use_drop:c
8277             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8278         }
8279     }
8280     \bool_set_false:N \g_@@_rotate_c_bool
8281 }

8282 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
8283 {
8284     \dim_gset:Nn \g_@@_blocks_ht_dim
8285     {
8286         \dim_max:nn
8287         \g_@@_blocks_ht_dim
8288         {
8289             \box_ht:c
8290             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8291         }

```

```

8292     }
8293     \dim_gset:Nn \g_@@_blocks_dp_dim
8294     {
8295         \dim_max:nn
8296         \g_@@_blocks_dp_dim
8297         {
8298             \box_dp:c
8299             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8300         }
8301     }
8302 }

8303 \cs_new:Npn \@@_adjust_hpos_rotate:
8304 {
8305     \bool_if:NT \g_@@_rotate_bool
8306     {
8307         \str_set:Ne \l_@@_hpos_block_str
8308         {
8309             \bool_if:NTF \g_@@_rotate_c_bool
8310             c
8311             {
8312                 \str_case:onF \l_@@_vpos_block_str
8313                 { b l B l t r T r }
8314                 {
8315                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
8316                     r
8317                     l
8318                 }
8319             }
8320         }
8321     }
8322 }
8323 \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*.

```

8324 \cs_new_protected:Npn \@@_rotate_box_of_block:
8325 {
8326     \box_grotate:cn
8327     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8328     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
8329     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
8330     {
8331         \vbox_gset_top:cn
8332         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8333         {
8334             \skip_vertical:n { 0.8 ex }
8335             \box_use:c
8336             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8337         }
8338     }
8339     \bool_if:NT \g_@@_rotate_c_bool
8340     {
8341         \hbox_gset:cn
8342         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8343         {
8344             \vbox_center:n
8345             {
8346                 \box_use:c
8347                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
8348             }
8349         }
8350     }
8351 }

```

```

8350     }
8351 }

```

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.

```

8352 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
8353 {
8354   \seq_gput_right:Ne \g_@@_blocks_seq
8355   {
8356     \l_tmpa_tl
8357     { \exp_not:n { #3 } }
8358     {
8359       \bool_if:NTF \l_@@_tabular_bool
8360       {
8361         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

8362   \@@_reset_arraystretch:
8363   \exp_not:n
8364   {
8365     \dim_zero:N \extrarowheight
8366     #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).

```

8367   \tag_if_active:T { \tag_stop:n { table } }
8368   \use:e
8369   {
8370     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
8371     { @ { } \l_@@_hpos_block_str @ { } }
8372   }
8373   #5
8374   \end { tabular }
8375 }
8376 }
8377 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

8378 {
8379 {

```

The following 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
8385     $ % $
8386     \use:e
8387     {
8388       \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8389       { @ { } \l_@@_hpos_block_str @ { } }
8390     }
8391     \@@_tuning_key_small:
8392     #5
8393     \end { array }

```

```

8394             $ % $
8395         }
8396     }
8397 }
8398 }
8399 }
8400 }
8401 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8402 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8403 {
8404     \seq_gput_right:Ne \g_@@_blocks_seq
8405     {
8406         \l_tmpa_tl
8407         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

8408         { { \exp_not:n { #4 #5 } } }
8409     }
8410 }
8411 \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`.

```

8412 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8413 {
8414     \seq_gput_right:Ne \g_@@_blocks_seq
8415     {
8416         \l_tmpa_tl
8417         { \exp_not:n { #3 } }
8418         { \exp_not:n { #4 #5 } }
8419     }
8420 }
8421 \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).

```

8422 \keys_define:nn { nicematrix / Block }
8423 {
8424     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8425     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8426     tikz .code:n =
8427         \IfPackageLoadedTF { tikz }
8428         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8429         { \@@_error:n { tikz~key~without~tikz } } ,
8430     tikz .value_required:n = true ,
8431     fill .code:n =
8432         \tl_set_rescan:Nnn
8433         \l_@@_fill_tl
8434         { \char_set_catcode_other:N ! }
8435         { #1 } ,
8436     fill .value_required:n = true ,

```

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

```

8437     opacity .tl_set:N = \l_@@_opacity_tl ,
8438     opacity .value_required:n = true ,
8439     draw .code:n =
8440         \tl_set_rescan:Nnn
8441         \l_@@_draw_tl

```

```

8442     { \char_set_catcode_other:N ! }
8443     { #1 } ,
8444     draw .default:n = default ,
8445     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8446     rounded-corners .default:n = 4 pt ,
8447     color .code:n =
8448       \@@_color:n { #1 }
8449       \tl_set_rescan:Nnn
8450         \l_@@_draw_tl
8451         { \char_set_catcode_other:N ! }
8452         { #1 } ,
8453     borders .clist_set:N = \l_@@_borders_clist ,
8454     borders .default:n = all ,
8455     hvlines .meta:n = { vlines , hlines } ,
8456     vlines .bool_set:N = \l_@@_vlines_block_bool ,
8457     hlines .bool_set:N = \l_@@_hlines_block_bool ,
8458     rules/width .dim_set:N = \arrayrulewidth ,
8459     rules/color .tl_set:N = \l_@@_rules_color_tl ,
8460     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8461     line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8462     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8463       \bool_set_true:N \l_@@_p_block_bool ,
8464     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8465     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8466     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8467     L .code:n = \str_set:Nn \l_@@_hpos_block_str L
8468       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8469     R .code:n = \str_set:Nn \l_@@_hpos_block_str R
8470       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8471     C .code:n = \str_set:Nn \l_@@_hpos_block_str C
8472       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8473     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8474     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8475     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8476     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8477     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8478     m .value_forbidden:n = true ,
8479     v-center .meta:n = m ,
8480     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8481     p .value_forbidden:n = true ,
8482     name .str_set:N = \l_@@_block_name_str ,
8483     name .value_required:n = true ,
8484     respect-arraystretch .code:n =
8485       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8486     respect-arraystretch .value_forbidden:n = true ,
8487     transparent .bool_set:N = \l_@@_transparent_bool ,
8488     unknown .code:n =
8489       \@@_unknown_key:nn
8490       { nicematrix / Block }
8491       { Unknown-key-for-Block }
8492   }

```

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.

```

8493 \cs_new_protected:Npn \@@_draw_blocks:
8494 {
8495   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8496   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8497 }

```

```

8498 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8499 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8500 \int_zero:N \l_@@_last_row_int
8501 \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”).

```

8502 \int_compare:nNnTF { #3 } > { 98 }
8503 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8504 { \int_set:Nn \l_@@_last_row_int { #3 } }
8505 \int_compare:nNnTF { #4 } > { 98 }
8506 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8507 { \int_set:Nn \l_@@_last_col_int { #4 } }

8508 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8509 {
8510   \bool_lazy_and:nnTF
8511     \l_@@_preamble_bool
8512     {
8513       \int_compare_p:n
8514         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8515     }
8516     { \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 } }
8517     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8518 }
8519 {
8520   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8521   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8522   {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8523 \use:e
8524 {
8525   \@@_Block_v:nnnnnn
8526   { #1 }
8527   { #2 }
8528   { \int_use:N \l_@@_last_row_int }
8529   { \int_use:N \l_@@_last_col_int }
8530 }
8531 { #5 }
8532 { #6 }
8533 }
8534 }
8535 }

```

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.

```

8536 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8537 {

```

The group is for the keys.

```

8538 \group_begin:
8539 \int_compare:nNnT { #1 } = { #3 }
8540 { \str_set:Nn \l_@@_vpos_block_str { t } }
8541 \keys_set:nn { nicematrix / Block } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

8542   \bool_if:NT \l_@@_amp_in_blocks_bool
8543     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

8544   \bool_lazy_and:nnT
8545     \l_@@_vlines_block_bool
8546     { ! \l_@@_ampersand_bool }
8547     {
8548       \tl_gput_right:Ne \g_@@_rules_tl
8549       {
8550         \@@_vlines_block:nnnnnn
8551         { \dim_use:N \arrayrulewidth }
8552         { \l_@@_rules_color_tl }
8553         { #1 } { #2 } { #3 } { #4 }
8554       }
8555     }
8556   \bool_if:NT \l_@@_hlines_block_bool
8557     {
8558       \tl_gput_right:Ne \g_@@_rules_tl
8559       {
8560         \@@_hlines_block:nnnnnn
8561         { \dim_use:N \arrayrulewidth }
8562         { \l_@@_rules_color_tl }
8563         { #1 } { #2 } { #3 } { #4 }
8564       }
8565     }

```

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).

```

8566   \bool_if:NF \l_@@_transparent_bool
8567     {
8568       \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8569       { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8570     }

8571   \tl_if_empty:NF \l_@@_draw_tl
8572     {
8573       \tl_gput_right:Nn \g_@@_rules_tl
8574       {
8575         \@@_stroke_block:nnnnn

```

#5 are the options

```

8576       { #5 } { #1 } { #2 } { #3 } { #4 }
8577     }
8578   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8579   { { #1 } { #2 } { #3 } { #4 } }
8580 }

8581 \clist_if_empty:NF \l_@@_borders_clist
8582 {
8583   \tl_gput_right:Nn \g_nicematrix_code_after_tl
8584   {
8585     \@@_stroke_borders_block:nnnnn
8586     { #5 } { #1 } { #2 } { #3 } { #4 }
8587   }
8588 }

8589 \tl_if_empty:NF \l_@@_fill_tl
8590 {
8591   \@@_add_opacity_to_fill:
8592   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8593   {
8594     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8595     { #1 - #2 }

```

```

8596         { #3 - #4 }
8597         { \dim_use:N \l_@@_rounded_corners_dim }
8598     }
8599 }

8600 \seq_if_empty:NF \l_@@_tikz_seq
8601 {
8602     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8603     {
8604         \@@_block_tikz:nnnnn
8605         { \seq_use:Nn \l_@@_tikz_seq { , } }
8606         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8607     }
8608 }

8609 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8610 {
8611     \tl_gput_right:Ne \g_@@_rules_tl
8612     {
8613         \@@_draw_diagbox:nnnnnn
8614         { #1 } { #2 } { #3 } { #4 }
8615         { \exp_not:n { ##1 } }
8616         { \exp_not:n { ##2 } }
8617     }
8618 }

```

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.

```

8619 \pgfpicture
8620 \pgfrememberpicturepositiononpagetrue
8621 \pgf@relevantforpicturesizefalse
8622 \@@_qpoint:n { row - #1 }
8623 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8624 \@@_qpoint:n { col - #2 }
8625 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8626 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8627 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8628 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8629 \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.

```

8630 \@@_pgf_rect_node:nnnnn
8631 { \@@_env: - #1 - #2 - block }
8632 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8633 \str_if_empty:NF \l_@@_block_name_str
8634 {
8635   \pgfnodealias
8636   { \@@_env: - \l_@@_block_name_str }
8637   { \@@_env: - #1 - #2 - block }
8638   \str_if_empty:NF \l_@@_name_str
8639   {
8640     \pgfnodealias
8641     { \l_@@_name_str - \l_@@_block_name_str }
8642     { \@@_env: - #1 - #2 - block }
8643   }
8644 }

```

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.

```

8645 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8646 {
8647   \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.

```

8648 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8649 {

```

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.

```

8650 \cs_if_exist:cT
8651 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8652 {
8653   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8654   {
8655     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8656     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8657   }
8658 }
8659 }

```

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.

```

8660 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8661 {
8662   \@@_qpoint:n { col - #2 }
8663   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8664 }
8665 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8666 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8667 {
8668   \cs_if_exist:cT
8669   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8670   {
8671     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8672     {
8673       \pgfpointanchor
8674       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8675       { east }
8676       \dim_set:Nn \l_@@_tmpd_dim
8677       { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8678     }
8679   }

```

```

8680     }
8681     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8682     {
8683         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8684         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8685     }
8686     \@@_pgf_rect_node:nnnnn
8687     { \@@_env: - #1 - #2 - block - short }
8688     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8689 }

```

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.

```

8690     \bool_if:NT \l_@@_medium_nodes_bool
8691     {
8692         \@@_pgf_rect_node:nnn
8693         { \@@_env: - #1 - #2 - block - medium }
8694         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
8695         {
8696             \pgfpointanchor
8697             { \@@_env:
8698               - \int_use:N \l_@@_last_row_int
8699               - \int_use:N \l_@@_last_col_int - medium
8700             }
8701             { south~east }
8702         }
8703     }
8704     \endpgfpicture
8705

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8706     \bool_if:NTF \l_@@_ampersand_bool
8707     {
8708         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8709         \int_zero_new:N \l_@@_split_int
8710         \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).

```

8711     \int_set:Nn \l_@@_first_row_int { #1 }
8712     \int_set:Nn \l_@@_first_col_int { #2 }
8713     \int_set:Nn \l_@@_last_row_int { #3 }
8714     \int_set:Nn \l_@@_last_col_int { #4 }
8715
8716     \pgfpicture
8717     \pgfrememberpicturepositiononpagetrue
8718     \pgf@relevantforpicturesizefalse
8719     \@@_qpoint:n { row - #1 }
8720     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8721     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8722     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8723     \@@_qpoint:n { col - #2 }
8724     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8725     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8726     \dim_set:Nn \l_tmpb_dim
8727     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8728     \bool_lazy_or:nnT
8729     \l_@@_vlines_block_bool
8730     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8731     {
8732         \int_step_inline:nn { \l_@@_split_int - 1 }
8733         {
8734             \pgfpathmoveto

```

```

8734         {
8735             \pgfpoint
8736             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8737             \l_@@_tmpc_dim
8738         }
8739         \pgfpathlineto
8740         {
8741             \pgfpoint
8742             { \l_tmpa_dim + ##1 \l_tmpb_dim }
8743             \l_@@_tmpd_dim
8744         }
8745         \CT@arc@
8746         \pgfsetlinewidth { 1.1 \arrayrulewidth }
8747         \pgfsetrectcap
8748         \pgfusepathqstroke
8749     }
8750 }
8751 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8752 \int_zero_new:N \l_@@_split_i_int
8753 \str_if_eq:eeTF \l_@@_vpos_block_str T
8754 {
8755     \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8756     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8757 }
8758 {
8759     \str_if_eq:eeTF \l_@@_vpos_block_str B
8760     {
8761         \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8762         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8763     }
8764     {
8765         \bool_lazy_or:nnTF
8766         { \int_compare_p:nNn { #1 } = { #3 } }
8767         { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8768         {
8769             \@@_qpoint:n { row - #1 - base }
8770             \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8771         }
8772         {
8773             \str_if_eq:eeTF \l_@@_vpos_block_str b
8774             {
8775                 \@@_qpoint:n { row - #3 - base }
8776                 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8777             }
8778             {
8779                 \@@_qpoint:n { #1 - #2 - block }
8780                 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8781             }
8782         }
8783     }
8784 }
8785 \int_step_inline:nn \l_@@_split_int
8786 {
8787     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8788     \int_set:Nn \l_@@_split_i_int { ##1 }
8789     \dim_set:Nn \col@sep
8790     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8791     \pgftransformshift
8792     {
8793         \pgfpoint
8794         {
8795             \l_tmpa_dim + ##1 \l_tmpb_dim -

```

```

8796         \str_case:on \l_@@_hpos_block_str
8797         {
8798             l { \l_tmpb_dim + \col@sep}
8799             c { 0.5 \l_tmpb_dim }
8800             r { \col@sep }
8801         }
8802     }
8803     { \l_@@_tmpc_dim }
8804 }
8805 \pgfset { inner~sep = \c_zero_dim }
8806 \pgfnode
8807 { rectangle }
8808 {
8809     \str_if_eq:eeTF T \l_@@_vpos_block_str
8810     {
8811         \str_case:on \l_@@_hpos_block_str
8812         {
8813             l { north~west }
8814             c { north }
8815             r { north~east }
8816         }
8817     }
8818     {
8819         \str_if_eq:eeTF B \l_@@_vpos_block_str
8820         {
8821             \str_case:on \l_@@_hpos_block_str
8822             {
8823                 l { south~west }
8824                 c { south }
8825                 r { south~east }
8826             }
8827         }
8828         {
8829             \bool_lazy_any:nTF
8830             {
8831                 { \int_compare_p:nNn { #1 } = { #3 } }
8832                 { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8833                 { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8834             }
8835             {
8836                 \str_case:on \l_@@_hpos_block_str
8837                 {
8838                     l { base~west }
8839                     c { base }
8840                     r { base~east }
8841                 }
8842             }
8843             {
8844                 \str_case:on \l_@@_hpos_block_str
8845                 {
8846                     l { west }
8847                     c { center }
8848                     r { east }
8849                 }
8850             }
8851         }
8852     }
8853 }
8854 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8855 \group_end:
8856 }
8857 \endpgfpicture
8858 }

```

Now the case where there is no ampersand & in the content of the block.

```

8859 {
8860   \bool_if:NTF \l_@@_p_block_bool
8861   {

```

When the final user has used the key p, we have to compute the width.

```

8862     \pgfpicture
8863     \pgfrememberpicturepositiononpagetrue
8864     \pgf@relevantforpicturesizefalse
8865     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8866     {
8867       \@@_qpoint:n { col - #2 }
8868       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8869       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8870     }
8871     {
8872       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8873       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8874       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8875     }
8876     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8877   \endpgfpicture
8878   \hbox_set:Nn \l_@@_cell_box
8879   {
8880     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8881     { \g_tmpb_dim }
8882     \str_case:on \l_@@_hpos_block_str
8883     { c \centering r \raggedleft l \raggedright j { } }
8884     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8885     #6
8886     \end { minipage }
8887   }
8888 }
8889 {
8890   \hbox_set:Nn \l_@@_cell_box
8891   {
8892     \set@color
8893     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8894     #6
8895   }
8896 }
8897 \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.

```

8898   \pgfpicture
8899   \pgfrememberpicturepositiononpagetrue
8900   \pgf@relevantforpicturesizefalse
8901   \bool_lazy_any:nTF
8902   {
8903     { \str_if_empty_p:N \l_@@_vpos_block_str }
8904     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8905     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8906     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8907   }
8908   {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8909     \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.

```

8910     \bool_if:nT \g_@@_last_col_found_bool

```

```

8911     {
8912         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8913         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8914     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8915     \tl_set:Ne \l_tmpa_tl
8916     {
8917         \str_case:on \l_@@_vpos_block_str
8918         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8919         { } {
8920             \str_case:on \l_@@_hpos_block_str
8921             {
8922                 c { center }
8923                 l { west }
8924                 r { east }
8925                 j { center }
8926             }
8927         }
8928     c {
8929         \str_case:on \l_@@_hpos_block_str
8930         {
8931             c { center }
8932             l { west }
8933             r { east }
8934             j { center }
8935         }
8936     }
8937 }
8938 T {
8939     \str_case:on \l_@@_hpos_block_str
8940     {
8941         c { north }
8942         l { north-west }
8943         r { north-east }
8944         j { north }
8945     }
8946 }
8947 }
8948 B {
8949     \str_case:on \l_@@_hpos_block_str
8950     {
8951         c { south }
8952         l { south-west }
8953         r { south-east }
8954         j { south }
8955     }
8956 }
8957 }
8958 }
8959 }
8960 \pgftransformshift
8961 {
8962     \pgfpointanchor
8963     {
8964         \@@_env: - #1 - #2 - block
8965         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8966     }
8967     \l_tmpa_tl
8968 }
8969 \pgfset { inner~sep = \c_zero_dim }

```

```

8970     \pgfnode
8971     { rectangle }
8972     \l_tmpa_tl
8973     { \box_use_drop:N \l_@@_cell_box } { } { }
8974 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8975 {
8976     \pgfextracty \l_tmpa_dim
8977     {
8978         \@@_qpoint:n
8979         {
8980             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8981             - base
8982         }
8983     }
8984     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8985     \pgfpointanchor
8986     {
8987         \@@_env: - #1 - #2 - block
8988         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8989     }
8990     {
8991         \str_case:on \l_@@_hpos_block_str
8992         {
8993             c { center }
8994             l { west }
8995             r { east }
8996             j { center }
8997         }
8998     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8999     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
9000     \pgfset { inner~sep = \c_zero_dim }
9001     \pgfnode
9002     { rectangle }
9003     {
9004         \str_case:on \l_@@_hpos_block_str
9005         {
9006             c { base }
9007             l { base~west }
9008             r { base~east }
9009             j { base }
9010         }
9011     }
9012     { \box_use_drop:N \l_@@_cell_box } { } { }
9013 }
9014 \endpgfpicture
9015 }
9016 \group_end:
9017 }
9018 \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).

```

9019 \cs_new_protected:Npn \@@_add_opacity_to_fill:
9020 {
9021     \tl_if_empty:NF \l_@@_opacity_tl
9022     {

```

```

9023 \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
9024 {
9025   \tl_set:Ne \l_@@_fill_tl
9026   {
9027     [ opacity = \l_@@_opacity_tl ,
9028     \tl_tail:o \l_@@_fill_tl
9029   }
9030 }
9031 {
9032   \tl_set:Ne \l_@@_fill_tl
9033   { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
9034 }
9035 }
9036 }

```

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*.

```

9037 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
9038 {
9039   \group_begin:
9040   \tl_clear:N \l_@@_draw_tl
9041   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9042   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
9043   \tl_if_empty:NF \l_@@_draw_tl
9044   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

9045     \tl_if_eq:NnTF \l_@@_draw_tl { default }
9046     \CT@arc@
9047     { \@@_color:o \l_@@_draw_tl }
9048   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

9049 \pgfsetcornersarced
9050 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
9051 \int_compare:nNnF { #2 } > \c@iRow
9052 {
9053   \int_compare:nNnF { #3 } > \c@jCol
9054   {
9055     \@@_qpoint:n { row - #2 }
9056     \dim_set_eq:NN \l_tmpb_dim \pgf@y
9057     \@@_qpoint:n { col - #3 }
9058     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9059     \@@_qpoint:n
9060     { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
9061     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9062     \@@_qpoint:n
9063     { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
9064     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
9065     \pgfpathrectanglecorners
9066     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
9067     { \pgfpoint \pgf@x \l_tmpa_dim }
9068     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
9069     \pgfusepathqstroke
9070     { \pgfusepath { stroke } }
9071   }
9072 }
9073 \group_end:
9074 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

9075 \keys_define:nn { nicematrix / BlockStroke }
9076 {

```

```

9077     color .tl_set:N = \l_@@_draw_tl ,
9078     draw .code:n =
9079       \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
9080     draw .default:n = default ,
9081     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9082     rounded-corners .default:n = 4 pt ,
9083     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

9084     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9085   }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The second is th color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

9086 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9087 {
9088   \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).

```

9089   \dim_set:Nn \arrayrulewidth { #1 }
9090   \pgfsetlinewidth \arrayrulewidth
9091   \@@_set_CTarc:n { #2 }
9092   \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).

```

9093   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9094   {
9095     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9096     ||
9097     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9098     ||
9099     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9100     ||
9101     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9102   }

9103   \bool_if:NT \l_@@_fix_vertex_bool
9104   {
9105     \int_compare:nNnT { #4 } > 1
9106     {
9107       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9108       {
9109         { 1 }
9110         { 1 }
9111         { \int_use:N \c@iRow }
9112         { \int_eval:n { #4 -1 } }
9113         { }
9114       }
9115     }
9116     \int_compare:nNnT { #6 } < \c@jCol
9117     {
9118       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9119       {
9120         { 1 }
9121         { \int_eval:n { #6 + 1 } }
9122         { \int_use:N \c@iRow }
9123         { \int_use:N \c@jCol }
9124         { }

```

```

9125     }
9126   }
9127 }
9128 \int_set:Nn \l_@@_start_int { #3 }
9129 \int_set:Nn \l_@@_end_int { #5 }
9130 \int_step_inline:nnn { #4 } { #6 + 1 }
9131 {
9132   \int_set:Nn \l_@@_position_int { ##1 }
9133   \socket_use:n { nicematrix / draw-vrule }
9134 }
9135 \group_end:
9136 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

9137 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
9138 {
9139   \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).

```

9140   \dim_set:Nn \arrayrulewidth { #1 }
9141   \pgfsetlinewidth \arrayrulewidth
9142   \@@_set_CTarc:n { #2 }
9143   \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).

```

9144   \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
9145   {
9146     \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
9147     ||
9148     \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
9149     ||
9150     \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
9151     ||
9152     \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
9153   }
9154   \bool_if:NT \l_@@_fix_vertex_bool
9155   {
9156     \int_compare:nNnT { #3 } > 1
9157     {
9158       \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9159       {
9160         { 1 }
9161         { 1 }
9162         { \int_eval:n { #3 - 1 } }
9163         { \int_use:N \c@jCol }
9164       }
9165     }
9166   }
9167   \int_compare:nNnT { #5 } < \c@iRow
9168   {
9169     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
9170     {
9171       { \int_eval:n { #5 + 1 } }
9172       { 1 }
9173       { \int_use:N \c@iRow }
9174       { \int_use:N \c@jCol }
9175     }

```

```

9176     }
9177   }
9178 }
9179 \int_set:Nn \l_@@_start_int { #4 }
9180 \int_set:Nn \l_@@_end_int { #6 }
9181 \int_step_inline:nnn { #3 } { #5 + 1 }
9182 {
9183   \int_set:Nn \l_@@_position_int { ##1 }
9184   \socket_use:n { nicematrix / draw-hrule }
9185 }
9186 \group_end:
9187 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

9188 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
9189 {
9190   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
9191   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
9192
9193   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
9194   { \@@_error:n { borders~forbidden } }
9195   {
9196     \tl_clear_new:N \l_@@_borders_tikz_tl
9197     \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
9198     \tl_set:Nn \l_@@_tmpc_tl { #2 }
9199     \tl_set:Nn \l_@@_tmpd_tl { #3 }
9200     \tl_set:Nc \l_tmpa_tl { \int_eval:n { #4 + 1 } }
9201     \tl_set:Nc \l_tmpb_tl { \int_eval:n { #5 + 1 } }
9202     \@@_stroke_borders_block_i:
9203   }
9204 }
9205 \AtBeginDocument
9206 {
9207   \cs_new_protected:Npe \@@_stroke_borders_block_i:
9208   {
9209     \c_@@_pgfortikzpicture_tl
9210     \@@_stroke_borders_block_ii:
9211     \c_@@_endpgfortikzpicture_tl
9212   }
9213 }
9214 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
9215 {
9216   \pgfrememberpicturepositiononpagetrue
9217   \pgf@relevantforpicturesizefalse
9218   \CT@arc@
9219   \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).

```

9220   \bool_set_false:N \l_tmpa_bool
9221   \clist_if_in:NnT \l_@@_borders_clist { right }
9222   {
9223     \@@_stroke_vertical:n \l_tmpb_tl
9224     \bool_set_true:N \l_tmpa_bool
9225   }
9226   \clist_if_in:NnT \l_@@_borders_clist { left }
9227   {
9228     \@@_stroke_vertical:n \l_@@_tmpd_tl
9229     \bool_set_true:N \l_tmpa_bool
9230   }
9231   \clist_if_in:NnT \l_@@_borders_clist { bottom }

```

```

9232     {
9233       \@@_stroke_horizontal:n \l_tmpa_tl
9234       \bool_set_true:N \l_tmpa_bool
9235     }
9236   \clist_if_in:NnT \l_@@_borders_clist { top }
9237   {
9238     \@@_stroke_horizontal:n \l_@@_tmpc_tl
9239     \bool_set_true:N \l_tmpa_bool
9240   }
9241   \bool_if:NF \l_tmpa_bool
9242   {
9243     \@@_stroke_vertical:n \l_tmpb_tl
9244     \@@_stroke_vertical:n \l_@@_tmpd_tl
9245     \@@_stroke_horizontal:n \l_tmpa_tl
9246     \@@_stroke_horizontal:n \l_@@_tmpc_tl
9247   }
9248 }
9249 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
9250 {
9251   tikz .code:n =
9252     \cs_if_exist:NTF \tikzpicture
9253     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
9254     { \@@_error:n { tikz~in~borders~without~tikz } } ,
9255   tikz .value_required:n = true ,
9256   dashed .meta:n = { tikz = dashed } ,
9257   top .code:n = ,
9258   bottom .code:n = ,
9259   left .code:n = ,
9260   right .code:n = ,
9261   all .code:n = ,
9262   unknown .code:n = \@@_error:n { bad~border }
9263 }

```

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).

```

9264 \cs_new_protected:Npn \@@_stroke_vertical:n #1
9265 {
9266   \@@_qpoint:n \l_@@_tmpc_tl
9267   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9268   \@@_qpoint:n \l_tmpa_tl
9269   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
9270   \@@_qpoint:n { #1 }
9271   \tl_if_empty:NTF \l_@@_borders_tikz_tl
9272   {
9273     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
9274     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
9275     \pgfusepath{stroke}
9276   }
9277   {
9278     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9279     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
9280   }
9281 }

```

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).

```

9282 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
9283 {
9284   \@@_qpoint:n \l_@@_tmpd_tl
9285   \clist_if_in:NnTF \l_@@_borders_clist { left }
9286   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
9287   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
9288   \@@_qpoint:n \l_tmpb_tl

```

```

9289 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
9290 \@@_qpoint:n { #1 }
9291 \tl_if_empty:NTF \l_@@_borders_tikz_tl
9292 {
9293   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
9294   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9295   \pgfusepathqstroke
9296 }
9297 {
9298   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
9299   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
9300 }
9301 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

9302 \keys_define:nn { nicematrix / BlockBorders }
9303 {
9304   borders .clist_set:N = \l_@@_borders_clist ,
9305   borders .default:n = all ,
9306   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
9307   rounded-corners .default:n = 4 pt ,
9308   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

9309   line-width .dim_set:N = \l_@@_line_width_dim % deprecated
9310 }

```

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.

```

9311 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
9312 {
9313   \begin { tikzpicture }
9314   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

9315   \clist_map_inline:nn { #1 }
9316   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

9317   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
9318   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
9319   (
9320     [
9321       xshift = \dim_use:N \l_@@_xoffset_dim ,
9322       yshift = - \dim_use:N \l_@@_yoffset_dim
9323     ]
9324     #2 -| #3
9325   )
9326   rectangle
9327   (
9328     [
9329       xshift = - \dim_use:N \l_@@_xoffset_dim ,
9330       yshift = \dim_use:N \l_@@_yoffset_dim
9331     ]
9332     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
9333   ) ;
9334 }
9335 \end { tikzpicture }

```

```

9336 }
9337 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

9338 \keys_define:nn { nicematrix / SpecialOffset }
9339 {
9340   xoffset .dim_set:N = \l_@@_xoffset_dim ,
9341   yoffset .dim_set:N = \l_@@_yoffset_dim ,
9342   offset .meta:n = { xoffset = #1 , yoffset = #1 }
9343 }

```

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.

```

9344 \cs_new_protected:Npn \@@_NullBlock:
9345 { \@@_collect_options:n { \@@_NullBlock_i: } }
9346 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
9347 { }

```

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.

```

9348 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
9349 {
9350   \tl_gput_right:Ne \g_@@_pre_code_before_tl
9351   {

```

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`).

```

9352   \@@_subcellcolor:nnnnnnn
9353   {
9354     \tl_if_blank:nTF { #1 }
9355     { { \exp_not:n { #2 } } }
9356     { [ #1 ] { \exp_not:n { #2 } } }
9357   }
9358   { \int_use:N \l_@@_first_row_int } % first row of the block
9359   { \int_use:N \l_@@_first_col_int } % first column of the block
9360   { \int_use:N \l_@@_last_row_int } % last row of the block
9361   { \int_use:N \l_@@_last_col_int } % last column of the block
9362   { \int_use:N \l_@@_split_int }
9363   { \int_use:N \l_@@_split_i_int }
9364 }
9365 \ignorespaces
9366 }

9367 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9368 {
9369   \@@_color_opacity: #1
9370   \pgfpicture
9371   \pgf@relevantforpicturesizefalse
9372   \@@_qpoint:n { col - #3 }
9373   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
9374   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
9375   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
9376   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
9377   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
9378   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9379   \@@_qpoint:n { row - #2 }
9380   \pgfpathrectanglecorners
9381   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
9382   { \pgfpoint \l_tmpb_dim \pgf@y }
9383   \pgfusepathqfill
9384   \endpgfpicture
9385 }

```

27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

9386 \keys_define:nn { nicematrix / Auto }
9387 {
9388   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9389   columns-type .value_required:n = true ,
9390   l .meta:n = { columns-type = l } ,
9391   r .meta:n = { columns-type = r } ,
9392   c .meta:n = { columns-type = c } ,
9393   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9394   delimiters / color .value_required:n = true ,
9395   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9396   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9397   delimiters .value_required:n = true ,
9398   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9399   rounded-corners .default:n = 4 pt
9400 }

9401 \NewDocumentCommand \AutoNiceMatrixWithDelims
9402 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9403 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

9404 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9405 {

```

The group is for the protection of the keys.

```

9406   \group_begin:
9407   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9408   \use:e
9409   {
9410     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9411     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9412     [ \exp_not:o \l_tmpa_tl ]
9413   }
9414   \int_if_zero:nT \l_@@_first_row_int
9415   {
9416     \int_if_zero:nT \l_@@_first_col_int { & }
9417     \prg_replicate:nn { #4 - 1 } { & }
9418     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9419   }
9420   \prg_replicate:nn { #3 }
9421   {
9422     \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).

```

9423     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9424     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9425   }
9426   \int_compare:nNnT \l_@@_last_row_int > { -2 }
9427   {
9428     \int_if_zero:nT \l_@@_first_col_int { & }
9429     \prg_replicate:nn { #4 - 1 } { & }
9430     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9431   }
9432   \end { NiceArrayWithDelims }
9433   \group_end:
9434 }

9435 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9436 {
9437   \cs_set_protected:cpn { #1 AutoNiceMatrix }

```

```

9438     {
9439         \bool_gset_true:N \g_@@_delims_bool
9440         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9441         \AutoNiceMatrixWithDelims { #2 } { #3 }
9442     }
9443 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

9444 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
9445 {
9446     \group_begin:
9447     \bool_gset_false:N \g_@@_delims_bool
9448     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9449     \group_end:
9450 }

```

28 The redefinition of the command `\dotfill`

```

9451 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9452 \cs_new_protected:Npn \@@_dotfill:
9453 {

```

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}`).

```

9454     \@@_old_dotfill:
9455     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9456 }

```

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`.

```

9457 \cs_new_protected:Npn \@@_dotfill_i:
9458 {
9459     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9460     { \@@_old_dotfill: }
9461 }

```

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.

```

9462 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9463 {
9464     \tl_gput_right:Ne \g_@@_rules_tl
9465     {
9466         \@@_draw_diagbox:nnnnnn
9467         { \int_use:N \c@iRow }
9468         { \int_use:N \c@jCol }
9469         { \int_use:N \c@iRow }
9470         { \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.

```

9471         { \g_@@_row_style_tl \exp_not:n { #1 } }
9472         { \g_@@_row_style_tl \exp_not:n { #2 } }
9473     }

```

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`.

```

9474     \seq_gput_right:N\g_@@_pos_of_blocks_seq
9475     {
9476       { \int_use:N \c@iRow }
9477       { \int_use:N \c@jCol }
9478       { \int_use:N \c@iRow }
9479       { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9480     { }
9481   }
9482 }

```

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.

```

9483 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9484 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9485   \begin { pgfscope }
9486   \@@_qpoint:n { row - #1 }
9487   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9488   \@@_qpoint:n { col - #2 }
9489   \dim_set_eq:NN \l_tmpb_dim \pgf@x
9490   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9491   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9492   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9493   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9494   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9495   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9496   {

```

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.

```

9497     \CT@arc@
9498     \pgfsetroundcap
9499     \pgfusepathqstroke
9500   }
9501   \pgfset { inner~sep = 1 pt }
9502   \pgfscope
9503   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9504   \pgfnode { rectangle } { south~west }
9505   {
9506     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

9507     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9508     \end { minipage }
9509   }
9510   { }
9511   { \pgfusepath { discard } } % mandatory
9512 \endpgfscope
9513 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9514 \pgfnode { rectangle } { north~east }
9515 {
9516   \begin { minipage } { 20 cm }
9517   \raggedleft
9518   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9519   \end { minipage }
9520 }
9521 { }

```

```

9522     { \pgfusepath { discard } } % mandatory
9523   \end { pgfscope }
9524 }

```

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`.

```

9525 \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 `\`.

```

9526 \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`.

```

9527 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9528 {
9529   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9530   \@@_CodeAfter_iv:n
9531 }

```

We catch the argument of the command `\end` (in `#1`).

```

9532 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9533 {

```

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.

```

9534   \str_if_eq:eeTF \@currenvir { #1 }
9535   { \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`.

```

9536   {
9537     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9538     \@@_CodeAfter_ii:n
9539   }
9540 }

```

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.

```

9541 \cs_new_protected:Npn \l_@@_delimiter:nnn #1 #2 #3
9542 {
9543   \pgfpicture
9544   \pgfrememberpicturepositiononpagetrue
9545   \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.

```

9546   \l_@@_qpoint:n { row - 1 }
9547   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - \arrayrulewidth }
9548   \l_@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9549   \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).

```

9550   \bool_if:nTF { #3 }
9551   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9552   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9553   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9554   {
9555     \cs_if_exist:cT
9556     { \pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
9557     {
9558       \pgfpointanchor
9559       { \l_@@_env: - ##1 - #2 }
9560       { \bool_if:nTF { #3 } { west } { east } }
9561       \dim_set:Nn \l_tmpa_dim
9562       {
9563         \bool_if:nTF { #3 }
9564         \dim_min:nn
9565         \dim_max:nn
9566         \l_tmpa_dim
9567         \pgf@x
9568       }
9569     }
9570   }

```

Now we can put the delimiter with a node of PGF.

```

9571   \pgfset { inner~sep = \c_zero_dim }
9572   \dim_zero:N \nulldelimiterspace
9573   \pgftransformshift
9574   {
9575     \pgfpoint
9576     \l_tmpa_dim
9577     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9578   }
9579   \pgfnode
9580   { rectangle }
9581   { \bool_if:nTF { #3 } { east } { west } }
9582   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9583   \nullfont
9584   $ % $
9585   \l_@@_color:o \l_@@_delimiters_color_tl
9586   \bool_if:nTF { #3 } { \left #1 } { \left . }
9587   \vcenter
9588   {
9589     \nullfont
9590     \hrule \@height
9591     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9592     \@depth \c_zero_dim
9593     \@width \c_zero_dim
9594   }

```

```

9595     \bool_if:nTF { #3 } { \right . } { \right #1 }
9596     $ % $
9597   }
9598   { }
9599   { }
9600   \endpgfpicture
9601 }

```

32 The command \SubMatrix

```

9602 \keys_define:nn { nicematrix / sub-matrix }
9603 {
9604   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9605   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9606   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9607   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9608   xshift .value_required:n = true ,
9609   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9610   delimiters / color .value_required:n = true ,
9611   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9612   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9613   hlines .default:n = all ,
9614   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9615   vlines .default:n = all ,
9616   hvlines .meta:n = { hlines, vlines } ,
9617   hvlines .value_forbidden:n = true
9618 }
9619 \keys_define:nn { nicematrix }
9620 {
9621   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9622   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9623   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9624   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9625 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

9626 \keys_define:nn { nicematrix / SubMatrix }
9627 {
9628   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9629   delimiters / color .value_required:n = true ,
9630   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9631   hlines .default:n = all ,
9632   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9633   vlines .default:n = all ,
9634   hvlines .meta:n = { hlines, vlines } ,
9635   hvlines .value_forbidden:n = true ,
9636   name .code:n =
9637     \tl_if_empty:nTF { #1 }
9638     { \@@_error:n { Invalid-name } }
9639     {
9640       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9641       {
9642         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9643         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9644         {
9645           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9646           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9647         }
9648       }
9649     }

```

```

9649         { \@@_error:n { Invalid-name } }
9650     } ,
9651     name .value_required:n = true ,
9652     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9653     rules .value_required:n = true ,
9654     code .tl_set:N = \l_@@_code_tl ,
9655     code .value_required:n = true ,
9656     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9657 }

9658 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9659 {
9660     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9661     {
9662         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9663         [
9664             delimiters / color = \l_@@_delimiters_color_tl ,
9665             hlines = \l_@@_submatrix_hlines_clist ,
9666             vlines = \l_@@_submatrix_vlines_clist ,
9667             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9668             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9669             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9670             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9671             #5
9672         ]
9673     }
9674     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9675     \ignorespaces
9676 }

9677 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9678 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9679 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9680 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9681 {
9682     \seq_gput_right:Ne \g_@@_submatrix_seq
9683     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9684         { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9685         { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9686         { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9687         { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9688     }
9689 }

```

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).

```

9690 \NewDocumentCommand \@@_compute_i_j:nn
9691 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9692 { \@@_compute_i_j:nnnn #1 #2 }

9693 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9694 {
9695     \def \l_@@_first_i_tl { #1 }
9696     \def \l_@@_first_j_tl { #2 }
9697     \def \l_@@_last_i_tl { #3 }
9698     \def \l_@@_last_j_tl { #4 }
9699     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9700     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9701     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9702     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9703     \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

9704      { \tl_set:NV \l_@@_last_i_tl \c{iRow }
9705      \tl_if_eq:NnT \l_@@_last_j_tl { last }
9706      { \tl_set:NV \l_@@_last_j_tl \c{jCol }
9707    }

```

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`.

```

9708 \AtBeginDocument
9709 {
9710   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9711   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9712     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9713 }
9714 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9715 {
9716   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9717   \@@_compute_i_j:nn { #2 } { #3 }
9718   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
9719     { \def \arraystretch { 1 } }
9720   \bool_lazy_or:nnTF
9721     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9722     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9723     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9724   {
9725     \str_clear_new:N \l_@@_submatrix_name_str
9726     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9727     \pgfpicture
9728     \pgfrememberpicturepositiononpagetrue
9729     \pgf@relevantforpicturesizefalse
9730     \pgfset { inner~sep = \c_zero_dim }
9731     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9732     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9733   \bool_if:NTF \l_@@_submatrix_slim_bool
9734     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9735     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9736   {
9737     \cs_if_exist:cT
9738       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9739     {
9740       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9741       \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9742         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9743     }
9744   \cs_if_exist:cT

```

```

9745         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9746         {
9747             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9748             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9749             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9750         }
9751     }
9752     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9753     { \@@_impossible_delimiter:n { left } }
9754     {
9755         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9756         { \@@_impossible_delimiter:n { right } }
9757         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9758     }
9759     \endpgfpicture
9760 }
9761 \group_end:
9762 \ignorespaces
9763 }

```

The argument of the following command will be provided by curryfication.

```

9764 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9765 {
9766     \bool_if:NTF \l_@@_no_cell_nodes_bool
9767     { \@@_error:n { Impossible-SubMatrix~no~cell-nodes } }
9768     { \@@_error:nn { Impossible-SubMatrix } { #1 } }
9769 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9770 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9771 {
9772     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9773     \dim_set:Nn \l_@@_y_initial_dim
9774     {
9775         \fp_to_dim:n
9776         {
9777             \pgf@y
9778             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9779             - \arrayrulewidth
9780         }
9781     }
9782     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9783     \dim_set:Nn \l_@@_y_final_dim
9784     {
9785         \fp_to_dim:n
9786         {
9787             \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch
9788             - \arrayrulewidth
9789         }
9790     }
9791     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9792     {
9793         \cs_if_exist:cT
9794         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9795         {
9796             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9797             \dim_set:Nn \l_@@_y_initial_dim
9798             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9799         }
9800         \cs_if_exist:cT
9801         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9802         {

```

```

9803         \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
9804         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9805         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9806     }
9807 }
9808 \dim_set:Nn \l_tmpa_dim
9809 {
9810     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9811     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9812 }
9813 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9814 \group_begin:
9815 \pgfsetlinewidth { 1.1 \arrayrulewidth }
9816 \l_@@_set_CTarc:o \l_@@_rules_color_tl
9817 \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`.

```

9818 \seq_map_inline:Nn \g_@@_cols_vlism_seq
9819 {
9820     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9821     {
9822         \int_compare:nNnT
9823         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9824         {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9825         \l_@@_qpoint:n { col - ##1 }
9826         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9827         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9828         \pgfusepathqstroke
9829     }
9830 }
9831 }

```

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.

```

9832 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9833 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9834 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9835 {
9836     \bool_lazy_and:nnTF
9837     { \int_compare_p:nNn { ##1 } > \c_zero_int }
9838     {
9839         \int_compare_p:nNn
9840         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9841     {
9842         \l_@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9843         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9844         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9845         \pgfusepathqstroke
9846     }
9847     { \l_@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9848 }

```

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.

```

9849 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9850 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9851 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }

```

```

9852 {
9853   \bool_lazy_and:nnTF
9854   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9855   {
9856     \int_compare_p:nNn
9857     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9858   {
9859     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect \l_tmpa_dim and \l_tmpb_dim.

```

9860   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
9861   \dim_set:Nn \l_tmpa_dim
9862   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9863   \str_case:nn { #1 }
9864   {
9865     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9866     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9867     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9868     }
9869     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in \l_tmpb_dim the *x*-value of the right end of the rule.

```

9870   \dim_set:Nn \l_tmpb_dim
9871   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9872   \str_case:nn { #2 }
9873   {
9874     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9875     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9876     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9877   }
9878   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9879   \pgfusepathqstroke
9880   \group_end:
9881 }
9882 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9883 }

```

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).

```

9884   \str_if_empty:NF \l_@@_submatrix_name_str
9885   {
9886     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9887     \l_@@_x_initial_dim \l_@@_y_initial_dim
9888     \l_@@_x_final_dim \l_@@_y_final_dim
9889   }
9890   \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.

```

9891   \begin { pgfscope }
9892   \pgftransformshift
9893   {
9894     \pgfpoint
9895     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9896     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9897   }
9898   \str_if_empty:NTF \l_@@_submatrix_name_str
9899   { \@@_node_left:nn #1 { } }
9900   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9901   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9902 \pgftransformshift
9903 {
9904   \pgfpoint
9905     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9906     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9907 }
9908 \str_if_empty:NTF \l_@@_submatrix_name_str
9909 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9910 {
9911   \@@_node_right:nnnn #2
9912   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9913 }

```

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`.

```

9914 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9915 \flag_clear_new:N \l_@@_code_flag
9916 \l_@@_code_tl
9917 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the *i* and *j* in specifications of nodes of the forms *i-j*, *row-i*, *col-j* and *i-|j* refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9918 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```

9919 \cs_new:Npn \@@_pgfpointanchor:n #1
9920 { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }

```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```

9921 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9922 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9923 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9924 {

```

The command `\str_if_empty:NTF` is “fully expandable”.

```

9925 \str_if_empty:NTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9926 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9927 { \@@_pgfpointanchor_ii:n { #1 } }
9928 }

```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```

9929 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9930 { \@@_pgfpointanchor_ii:n { #1 } }

```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
9931 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
9932 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9933 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9934 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9935 { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
9936 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9937 }
```

The following function is for the case when the name contains an hyphen.

```
9938 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9939 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9940 \@@_env:
9941 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9942 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9943 }
```

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.

```
9944 \tl_const:Nn \c_@@_integers_alist_tl
9945 {
9946 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9947 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9948 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9949 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9950 }
```

```
9951 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9952 {
```

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`.

```
9953 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9954 {
9955 \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9956 \@@_env: -
9957 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9958 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9959 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9960 }
```

```

9961 {
9962   \str_if_eq:eeTF { #1 } { last }
9963   {
9964     \flag_raise:N \l_@@_code_flag
9965     \@@_env: -
9966     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9967       { \int_eval:n { \l_@@_last_i_tl + 1 } }
9968       { \int_eval:n { \l_@@_last_j_tl + 1 } }
9969   }
9970   { #1 }
9971 }
9972 }

```

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`).

```

9973 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9974 {
9975   \pgfnode
9976   { rectangle }
9977   { east }
9978   {
9979     \nullfont
9980     $ % $
9981     \@@_color:o \l_@@_delimiters_color_tl
9982     \left #1
9983     \vcenter
9984     {
9985       \nullfont
9986       \hrule \@height \l_tmpa_dim
9987         \depth \c_zero_dim
9988         \width \c_zero_dim
9989     }
9990     \right .
9991     $ % $
9992   }
9993   { #2 }
9994   { }
9995 }

```

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.

```

9996 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9997 {
9998   \pgfnode
9999   { rectangle }
10000   { west }
10001   {
10002     \nullfont
10003     $ % $
10004     \colorlet { current-color } { . }
10005     \@@_color:o \l_@@_delimiters_color_tl
10006     \left .
10007     \vcenter
10008     {
10009       \nullfont
10010       \hrule \@height \l_tmpa_dim
10011         \depth \c_zero_dim
10012         \width \c_zero_dim
10013     }
10014     \right #1
10015     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }

```

```

10016      ^ { \color { current-color } \smash { #4 } }
10017      $ % $
10018    }
10019    { #2 }
10020    { }
10021  }

```

33 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

10022 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
10023 {
10024   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
10025   \ignorespaces
10026 }

10027 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
10028 {
10029   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
10030   \ignorespaces
10031 }

10032 \keys_define:nn { nicematrix / Brace }
10033 {
10034   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
10035   left-shorten .value_forbidden:n = true ,
10036   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
10037   right-shorten .value_forbidden:n = true ,
10038   shorten .meta:n = { left-shorten , right-shorten } ,
10039   shorten .value_forbidden:n = true ,
10040   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
10041   color .tl_set:N = \l_tmpa_tl ,
10042   color .value_required:n = true ,
10043   unknown .code:n =
10044     \@@_unknown_key:nn
10045       { nicematrix / Brace }
10046       { Unknown-key-for-Brace }
10047 }

```

#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*.

```

10048 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
10049 {
10050   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

10051 \@@_compute_i_j:nn { #1 } { #2 }
10052 \bool_lazy_or:nnTF
10053   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
10054   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
10055   {
10056     \str_if_eq:eeTF { #5 } { under }
10057     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
10058     { \@@_error:nn { Construct-too-large } { \OverBrace } }
10059   }
10060   {
10061     \tl_clear:N \l_tmpa_tl
10062     \keys_set:nn { nicematrix / Brace } { #4 }

```

```

10063 \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
10064 \pgfpicture
10065 \pgfrememberpicturepositiononpagetrue
10066 \pgf@relevantforpicturesizefalse
10067 \bool_if:NT \l_@@_brace_left_shorten_bool
10068 {
10069   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
10070   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10071   {
10072     \cs_if_exist:cT
10073     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
10074     {
10075       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
10076
10077       \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
10078       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
10079     }
10080   }
10081 }
10082 \bool_lazy_or:nnT
10083 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
10084 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
10085 {
10086   \@@_qpoint:n { col - \l_@@_first_j_tl }
10087   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
10088 }
10089 \bool_if:NT \l_@@_brace_right_shorten_bool
10090 {
10091   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
10092   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
10093   {
10094     \cs_if_exist:cT
10095     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
10096     {
10097       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
10098       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
10099       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
10100     }
10101   }
10102 }
10103 \bool_lazy_or:nnT
10104 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
10105 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
10106 {
10107   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
10108   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
10109 }
10110 \pgfset { inner~sep = \c_zero_dim }
10111 \str_if_eq:eeTF { #5 } { under }
10112 { \@@_underbrace_i:n { #3 } }
10113 { \@@_overbrace_i:n { #3 } }
10114 \endpgfpicture
10115 }
10116 \group_end:
10117 }

```

The argument is the text to put above the brace.

```

10118 \cs_new_protected:Npn \@@_overbrace_i:n #1
10119 {
10120   \@@_qpoint:n { row - \l_@@_first_i_tl }
10121   \pgftransformshift
10122   {
10123     \pgfpoint
10124     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }

```

```

10125         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
10126     }
10127 \pgfnode
10128 { rectangle }
10129 { south }
10130 {
10131     \vtop
10132     {
10133         \group_begin:
10134         \everycr { }
10135         \halign
10136         {
10137             \hfil ## \hfil \crcr
10138             \bool_if:NTF \l_@@_tabular_bool
10139             { \begin { tabular } { c } #1 \end { tabular } }
10140             { $ \begin { array } { c } #1 \end { array } $ }
10141             \cr
10142             $ % $
10143             \overbrace
10144             {
10145                 \hbox_to_wd:nn
10146                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10147                 { }
10148             }
10149             $ % $
10150             \cr
10151         }
10152         \group_end:
10153     }
10154 }
10155 { }
10156 { }
10157 }

```

The argument is the text to put under the brace.

```

10158 \cs_new_protected:Npn \@@_underbrace_i:n #1
10159 {
10160     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
10161     \pgftransformshift
10162     {
10163         \pgfpoint
10164         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
10165         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
10166     }
10167 \pgfnode
10168 { rectangle }
10169 { north }
10170 {
10171     \group_begin:
10172     \everycr { }
10173     \vbox
10174     {
10175         \halign
10176         {
10177             \hfil ## \hfil \crcr
10178             $ % $
10179             \underbrace
10180             {
10181                 \hbox_to_wd:nn
10182                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
10183                 { }
10184             }
10185             $ % $
10186             \cr

```

```

10187         \bool_if:NTF \l_@@_tabular_bool
10188         { \begin { tabular } { c } #1 \end { tabular } }
10189         { $ \begin { array } { c } #1 \end { array } $ }
10190     \cr
10191 }
10192 }
10193 \group_end:
10194 }
10195 { }
10196 { }
10197 }

```

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`).

```

10198 \AddToHook { package / tikz / after }
10199 {
10200     \tikzset
10201     {
10202         nicematrix / brace / .style =
10203         {
10204             decoration = { brace , raise = -0.15 em } ,
10205             decorate ,
10206         } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

10207         nicematrix / mirrored-brace / .style =
10208         {
10209             nicematrix / brace ,
10210             decoration = mirror ,
10211         }
10212     }
10213 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

10214 \keys_define:nn { nicematrix / Hbrace }
10215 {
10216     color .code:n = ,
10217     horizontal-label .code:n = ,
10218     horizontal-labels .code:n = ,
10219     shorten .code:n = ,
10220     shorten-start .code:n = ,
10221     shorten-end .code:n = ,
10222     shorten+ .code:n = ,
10223     shorten-start+ .code:n = ,
10224     shorten-end+ .code:n = ,
10225     shorten~+ .code:n = ,
10226     shorten-start~+ .code:n = ,
10227     shorten-end~+ .code:n = ,
10228     brace-shift .code:n = ,
10229     brace-shift+ .code:n = ,
10230     brace-shift~+ .code:n = ,

```

```

10231     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
10232 }

```

Here we need an “fully expandable” command.

```

10233 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
10234 {
10235     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10236     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
10237     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
10238 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

10239 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
10240 {
10241     \int_compare:nNnTF \c@iRow < { 2 }
10242     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

10243     \str_if_eq:nnTF { #2 } { * }
10244     {
10245         \bool_set_true:N \l_@@_nullify_dots_bool
10246         \Ldots
10247         [
10248             line-style = nicematrix / brace ,
10249             #1 ,
10250             up =
10251             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10252         ]
10253     }
10254     {
10255         \Hdotsfor
10256         [
10257             line-style = nicematrix / brace ,
10258             #1 ,
10259             up =
10260             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10261         ]
10262         { #2 }
10263     }
10264 }
10265 {
10266     \str_if_eq:nnTF { #2 } { * }
10267     {
10268         \bool_set_true:N \l_@@_nullify_dots_bool
10269         \Ldots
10270         [
10271             line-style = nicematrix / mirrored-brace ,
10272             #1 ,
10273             down =
10274             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10275         ]
10276     }
10277     {
10278         \Hdotsfor
10279         [
10280             line-style = nicematrix / mirrored-brace ,
10281             #1 ,
10282             down =
10283             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10284         ]
10285         { #2 }
10286     }

```

```

10287     }
10288     \keys_set:nn { nicematrix / Hbrace } { #1 }
10289 }

10290 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
10291 {
10292     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
10293     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
10294     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
10295 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

10296 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
10297 {
10298     \int_compare:nNnTF \c@jCol < { 2 }
10299     {
10300         \str_if_eq:nnTF { #2 } { * }
10301         {
10302             \bool_set_true:N \l_@@_nullify_dots_bool
10303             \Vdots
10304             [
10305                 Vbrace ,
10306                 line-style = nicematrix / mirrored-brace ,
10307                 #1 ,
10308                 down =
10309                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10310             ]
10311         }
10312         {
10313             \Vdotsfor
10314             [
10315                 Vbrace ,
10316                 line-style = nicematrix / mirrored-brace ,
10317                 #1 ,
10318                 down =
10319                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10320             ]
10321             { #2 }
10322         }
10323     }
10324     {
10325         \str_if_eq:nnTF { #2 } { * }
10326         {
10327             \bool_set_true:N \l_@@_nullify_dots_bool
10328             \Vdots
10329             [
10330                 Vbrace ,
10331                 line-style = nicematrix / brace ,
10332                 #1 ,
10333                 up =
10334                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10335             ]
10336         }
10337         {
10338             \Vdotsfor
10339             [
10340                 Vbrace ,
10341                 line-style = nicematrix / brace ,
10342                 #1 ,
10343                 up =
10344                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
10345             ]

```

```

10346         { #2 }
10347     }
10348 }
10349 \keys_set:nn { nicematrix / Hbrace } { #1 }
10350 }

```

35 The command TikzEveryCell

```

10351 \bool_new:N \l_@@_not_empty_bool
10352 \bool_new:N \l_@@_empty_bool
10353
10354 \keys_define:nn { nicematrix / TikzEveryCell }
10355 {
10356     not-empty .code:n =
10357         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10358         { \bool_set_true:N \l_@@_not_empty_bool }
10359         { \@@_error:n { detection-of-empty-cells } } ,
10360     not-empty .value_forbidden:n = true ,
10361     empty .code:n =
10362         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
10363         { \bool_set_true:N \l_@@_empty_bool }
10364         { \@@_error:n { detection-of-empty-cells } } ,
10365     empty .value_forbidden:n = true ,
10366     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
10367 }
10368
10369
10370 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
10371 {
10372     \IfPackageLoadedTF { tikz }
10373     {
10374         \group_begin:
10375         \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.

```

10376         \tl_set:Nn \l_tmpa_tl { { #2 } }
10377         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
10378         { \@@_for_a_block:nnnnn ##1 }
10379         \@@_all_the_cells:
10380         \group_end:
10381     }
10382     { \@@_error:n { TikzEveryCell-without-tikz } }
10383 }
10384
10385
10386 \cs_new_protected:Nn \@@_all_the_cells:
10387 {
10388     \int_step_inline:nn \c@iRow
10389     {
10390         \int_step_inline:nn \c@jCol
10391         {
10392             \cs_if_exist:cF { cell - ##1 - #####1 }
10393             {
10394                 \@@_if_in_corner:nF { ##1 - #####1 }
10395                 {
10396                     \bool_set_false:N \l_tmpa_bool
10397                     \cs_if_exist:cTF
10398                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10399                     {

```

```

10400             \bool_if:NF \l_@@_empty_bool
10401             { \bool_set_true:N \l_tmpa_bool }
10402         }
10403     {
10404         \bool_if:NF \l_@@_not_empty_bool
10405         { \bool_set_true:N \l_tmpa_bool }
10406     }
10407     \bool_if:NT \l_tmpa_bool
10408     {
10409         \@@_block_tikz:nnnnn
10410         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10411     }
10412 }
10413 }
10414 }
10415 }
10416 }
10417
10418 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10419 {
10420     \bool_if:NF \l_@@_empty_bool
10421     {
10422         \@@_block_tikz:nnnnn
10423         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10424     }
10425     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10426 }
10427
10428 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10429 {
10430     \int_step_inline:nnn { #1 } { #3 }
10431     {
10432         \int_step_inline:nnn { #2 } { #4 }
10433         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10434     }
10435 }

```

36 The key draw-trees-in-col

```

10436 \cs_new_protected:Npn \@@_draw_trees:
10437 {
10438     \bool_if:NTF \l_@@_no_cell_nodes_bool
10439     { \@@_error:n { draw-trees-with-no-cell-nodes } }
10440     \@@_draw_trees_i:
10441 }
10442 \cs_new_protected:Npn \@@_draw_trees_i:
10443 {
10444     \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10445     \dim_zero_new:N \l_@@_em_dim
10446     \dim_set:Nn \l_@@_em_dim { 1 em }
10447     \dim_zero_new:N \l_@@_ex_dim
10448     \dim_set:Nn \l_@@_ex_dim { 1 ex }
10449     \pgfpicture
10450     \pgfrememberpicturepositiononpagetrue
10451     \pgf@relevantforpicturesizefalse
10452     \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10453     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10454     \@@_color:o \l_@@_trees_color_tl
10455     \pgfsetcornersarced
10456     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10457     \clist_map_function:NN \g_@@_col_with_trees_clist
10458     \@@_draw_trees_in_col:n

```

```

10459 \clist_gclear:N \g_@@_col_with_trees_clist
10460 \endpgfpicture
10461 }

```

```

10462 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10463 {

```

The argument is provided by curryfication.

```

10464 \int_compare:nNnTF { #1 } > \c@jCol
10465 { \@@_error:nn { Col-outside~tabular~in~trees } }
10466 {
10467   \int_compare:nNnTF { #1 } = \c@jCol
10468   { \@@_error:nn { Last~col~in~trees } }
10469   \@@_draw_trees_in_col_i:n
10470 }
10471 { #1 }
10472 }

```

```

10473 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10474 {
10475   \int_set:Nn \l_tmpa_int { 1 }
10476   \int_step_inline:nn { \c@iRow + 1 }
10477   {
10478     \cs_if_exist:cT
10479     { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10480     {
10481       \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10482       {

```

Now, you will, potentially, draw a tree.

```

10483       \@@_draw_tree:nee
10484       { #1 }
10485       { \int_use:N \l_tmpa_int }
10486       { \int_eval:n { ##1 - 1 } }
10487     }
10488     \int_set:Nn \l_tmpa_int { ##1 }
10489   }
10490 }
10491 \int_compare:nNnT \c@iRow > \l_tmpa_int
10492 {
10493   \@@_draw_tree:nee
10494   { #1 }
10495   { \int_use:N \l_tmpa_int }
10496   { \int_eval:n { \c@iRow } }
10497 }
10498 }

```

#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.

```

10499 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10500 {

```

\l_tmpa_dim will be the x -value of the vertical rule that we will draw.

```

10501 \pgfpointanchor { \@@_env: - #1 } { 5 }
10502 \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.

```

10503 \bool_set_false:N \l_tmpa_bool
10504 \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10505 {
10506   \cs_if_exist:cT
10507   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10508 {
10509   \pgfpointanchor
10510   { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10511   { base~west }
10512   \pgfpathmoveto
10513   {
10514     \pgfpoint
10515     { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10516     { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10517   }
10518   \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10519   \bool_if:NF \l_tmpa_bool
10520   {
10521     \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10522     \pgfpathlineto
10523     { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10524     \bool_set_true:N \l_tmpa_bool
10525   }
10526   \pgfusepath { stroke }
10527 }
10528 }
10529 }
10530 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

37 The key create-blocks-in-col

```

10531 \cs_new_protected:Npn \@@_create_blocks_in_col:
10532 {
10533   \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10534   \clist_map_inline:Nn \g_@@_cbic_clist
10535   {
10536     \cs_set:cpn
10537     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { \c@iRow + 1 } - ##1 }
10538     { rien }

```

`\l_tmpa_int` will be the first row of the block being created.

```

10539   \int_set:Nn \l_tmpa_int { 1 }
10540   \int_step_inline:nn { \c@iRow + 1 }
10541   {
10542     \bool_set_false:N \l_tmpa_bool
10543     \cs_if_exist:cT { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10544     { \bool_set_true:N \l_tmpa_bool }
10545     \clist_if_in:NnT \g_@@_false_rows_clist { #####1 }
10546     { \bool_set_true:N \l_tmpa_bool }
10547     \bool_if:NT \l_tmpa_bool
10548     {
10549       \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10550       {
10551         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10552         {
10553           { \int_use:N \l_tmpa_int }
10554           { ##1 }
10555           { \int_eval:n { #####1 - 1 } }
10556           { ##1 }
10557           { }
10558         }
10559       }
10560       \int_set:Nn \l_tmpa_int { #####1 }
10561     }
10562   }
10563 }
10564 \clist_gclear:N \g_@@_cbic_clist

```

```
10565 }
```

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.

```
10566 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10567 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10568 \NewDocumentCommand \@@_ShowCellNames { }
10569 {
10570   \bool_if:NT \l_@@_in_code_after_bool
10571   {
10572     \pgfpicture
10573     \pgfrememberpicturepositiononpagetrue
10574     \pgf@relevantforpicturesizefalse
10575     \pgfpathrectanglecorners
10576       { \@@_qpoint:n { 1 } }
10577       { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10578     \pgfsetfillopacity { 0.75 }
10579     \pgfsetfillcolor { white }
10580     \pgfusepathqfill
10581     \endpgfpicture
10582   }
10583   \dim_gzero_new:N \g_@@_tmpc_dim
10584   \dim_gzero_new:N \g_@@_tmpd_dim
10585   \dim_gzero_new:N \g_@@_tmpe_dim
10586   \int_step_inline:nn \c@iRow
10587   {
10588     \clist_if_in:NnF \g_@@_false_rows_clist { ##1 }
10589     { \@@_show_cell_names_one_row:n { ##1 } }
10590   }
10591 }

10592 \cs_new_protected:Npn \@@_show_cell_names_one_row:n #1
10593 {
10594   \bool_if:NTF \l_@@_in_code_after_bool
10595   {
10596     \pgfpicture
10597     \pgfrememberpicturepositiononpagetrue
10598     \pgf@relevantforpicturesizefalse
10599   }
10600   { \begin { pgfpicture } }
10601   \@@_qpoint:n { row - #1 }
10602   \dim_set_eq:NN \l_tmpa_dim \pgf@y
10603   \@@_qpoint:n { row - \int_eval:n { #1 + 1 } }
10604   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10605   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10606   \bool_if:NTF \l_@@_in_code_after_bool
10607   { \endpgfpicture }
10608   { \end { pgfpicture } }
10609   \int_step_tokens:nn \c@jCol
10610   { \@@_show_cell_names_one_cell:nn { #1 } }
10611 }
```

```

10612 \cs_new_protected:Npn \@@_show_cell_names_one_cell:nn #1 #2
10613 {
10614   \clist_if_in:NnF \g_@@_false_cols_clist { #2 }
10615   { \@@_show_cell_names_one_cell_i:nn { #1 } { #2 } }
10616 }
10617 \cs_new_protected:Npn \@@_show_cell_names_one_cell_i:nn #1 #2
10618 {
10619   \hbox_set:Nn \l_tmpa_box
10620   {
10621     \normalfont \Large \sffamily \bfseries
10622     \bool_if:NTF \l_@@_in_code_after_bool
10623     { \color { red } }
10624     { \color { red ! 50 } }
10625     #1 - #2
10626   }
10627   \bool_if:NTF \l_@@_in_code_after_bool
10628   {
10629     \pgfpicture
10630     \pgfrememberpicturepositiononpagetrue
10631     \pgf@relevantforpicturesizefalse
10632   }
10633   { \begin { pgfpicture } }
10634   \@@_qpoint:n { col - #2 }
10635   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10636   \@@_qpoint:n { col - \int_eval:n { #2 + 1 } }
10637   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10638   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10639   \bool_if:NTF \l_@@_in_code_after_bool
10640   { \endpgfpicture }
10641   { \end { pgfpicture } }
10642   \fp_set:Nn \l_tmpa_fp
10643   {
10644     \fp_min:nn
10645     {
10646       \fp_min:nn
10647       { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10648       { \dim_ratio:nn \g_@@_tmpe_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10649     }
10650     { 1.0 }
10651   }
10652   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10653   \pgfpicture
10654   \pgfrememberpicturepositiononpagetrue
10655   \pgf@relevantforpicturesizefalse
10656   \pgftransformshift
10657   {
10658     \pgfpoint
10659     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10660     \g_@@_tmpa_dim
10661   }
10662   \pgfnode
10663   { rectangle }
10664   { center }
10665   { \box_use:N \l_tmpa_box }
10666   { }
10667   { }
10668   \endpgfpicture
10669 }

```

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.

```
10670 \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.

```
10671 \bool_new:N \g_@@_footnote_bool
10672 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10673 {
10674   You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
10675   but~that~key~is~unknown. \\\
10676   It~will~be~ignored. \\\
10677   For~a~list~of~the~available~keys,~type~H~<return>.
10678 }
10679 {
10680   The~available~keys~are~(in~alphabetic~order):~
10681   footnote,~
10682   footnotehyper,~
10683   messages~for~Overleaf,~
10684   renew~dots~and~
10685   renew~matrix.
10686 }
10687 \keys_define:nn { nicematrix }
10688 {
10689   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
10690   renew~dots .value_forbidden:n = true ,
10691   renew~matrix .code:n = \@@_renew_matrix: ,
10692   renew~matrix .value_forbidden:n = true ,
10693   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10694   footnote .bool_set:N = \g_@@_footnote_bool ,
10695   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10696   unknown .code:n = \@@_error:n { Unknown~key~for~package }
10697 }
10698 \ProcessKeyOptions
10699 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10700 {
10701   You~can't~use~the~option~'footnote'~because~the~package~
10702   footnotehyper~has~already~been~loaded.~
10703   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10704   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10705   of~the~package~footnotehyper.\\
10706   The~package~footnote~won't~be~loaded.
10707 }
10708 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10709 {
10710   You~can't~use~the~option~'footnotehyper'~because~the~package~
10711   footnote~has~already~been~loaded.~
10712   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10713   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10714   of~the~package~footnote.\\
10715   The~package~footnotehyper~won't~be~loaded.
10716 }
```

```

10717 \bool_if:NT \g_@@_footnote_bool
10718 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10719   \IfClassLoadedTF { beamer }
10720   { \bool_set_false:N \g_@@_footnote_bool }
10721   {
10722     \IfPackageLoadedTF { footnotehyper }
10723     { \@@_error:n { footnote~with~footnotehyper~package } }
10724     { \usepackage { footnote } }
10725   }
10726 }
10727 \bool_if:NT \g_@@_footnotehyper_bool
10728 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10729   \IfClassLoadedTF { beamer }
10730   { \bool_set_false:N \g_@@_footnote_bool }
10731   {
10732     \IfPackageLoadedTF { footnote }
10733     { \@@_error:n { footnotehyper~with~footnote~package } }
10734     { \usepackage { footnotehyper } }
10735   }
10736   \bool_set_true:N \g_@@_footnote_bool
10737 }

```

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.

```

10738 \IfPackageLoadedF { underscore }
10739 {
10740   \AddToHook { package / underscore / after }
10741   { \@@_error:n { underscore~after~nicematrix } }
10742 }

```

41 Compatibility with threeparttable

```

10743 \AddToHook { package / threeparttable / after }
10744 {
10745   \AddToHook { env / threeparttable / begin }
10746   {
10747     \TPT@hookin { NiceTabular }
10748     \TPT@hookin { NiceTabular* }
10749     \TPT@hookin { NiceTabularX }
10750   }
10751 }

```

42 Gestion of the errors

42.1 Security in the CodeBefore and the CodeAfter

```
10752 \cs_new_protected:Npn \@@_security_font_commands:
10753 {
10754   \cs_set_eq:NN \@@_old_small: \small
10755   \cs_set_eq:NN \@@_old_footnotesize: \footnotesize
10756   \cs_set_eq:NN \@@_old_scriptsize: \scriptsize
10757   \cs_set_eq:NN \@@_old_tiny: \tiny
10758   \cs_set_eq:NN \small \@@_small:
10759   \cs_set_eq:NN \footnotesize \@@_footnotesize:
10760   \cs_set_eq:NN \scriptsize \@@_scriptsize:
10761   \cs_set_eq:NN \tiny \@@_tiny:
10762   \everyhbox
10763   {
10764     \cs_set_eq:NN \small \@@_old_small:
10765     \cs_set_eq:NN \footnotesize \@@_old_footnotesize:
10766     \cs_set_eq:NN \scriptsize \@@_old_scriptsize:
10767     \cs_set_eq:NN \tiny \@@_old_tiny:
10768   }
10769 }

10770 \cs_set_protected:Npn \@@_small: { \@@_font_command_error:N \small }
10771 \cs_set_protected:Npn \@@_footnotesize: { \@@_font_command_error:N \footnotesize }
10772 \cs_set_protected:Npn \@@_scriptsize: { \@@_font_command_error:N \scriptsize }
10773 \cs_set_protected:Npn \@@_tiny: { \@@_font_command_error:N \tiny }

10774 \cs_new_protected:Npn \@@_font_command_error:N #1
10775 { \@@_error:nn { font-command } { #1 } }

10776 \@@_msg_new:nn { font-command }
10777 {
10778   Font~command~not~allowed.\\
10779   You~can't~use~the~command~#1~directly~in~the~
10780   \bool_if:NTF \l_@@_in_code_after_bool
10781   { \token_to_str:N \CodeAfter}
10782   { \token_to_str:N \CodeBefore}.~Your~command~will~be~ignored.
10783 }
```

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.

```
10784 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10785 {
10786   \str_if_eq:eeTF
10787   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10788   { + }
10789   {
10790     \str_set:Ne \l_tmpa_str
10791     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10792     \bool_set_false:N \l_tmpa_bool
10793     \clist_map_inline:nn { #1 }
10794     {
10795       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10796       {
10797         \@@_error:n { key~without~+~exists }
10798         \bool_set_true:N \l_tmpa_bool
10799         \clist_map_break:
10800       }
10801     }
10802   }
```

```

10802     \bool_if:NF \l_tmpa_bool
10803     {
10804         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10805         \@@_unknown_key_i:nn { #1 } { #2 }
10806     }
10807 }
10808 { \@@_unknown_key_i:nn { #1 } { #2 } }
10809 }

```

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.

```

10810 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10811 {
10812     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10813     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10814     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10815     \bool_set_false:N \l_tmpa_bool
10816     \clist_map_inline:nn { #1 }
10817     {
10818         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10819         {
10820             \@@_error:n { key~with~normal~form~exists }
10821             \bool_set_true:N \l_tmpa_bool
10822             \clist_map_break:
10823         }
10824     }
10825     \bool_if:NF \l_tmpa_bool
10826     {
10827         \@@_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).

```

10828     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10829     { \msg_info:nn { nicematrix } { #2~+ } }
10830 }
10831 }
10832 \@@_msg_new:nn { key~without~+~exists }
10833 {
10834     Problem~with~a~key~\l
10835     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10836     additive~syntax~(with~+=).~It~will~be~ignored.
10837 }
10838 \@@_msg_new:nn { key~with~normal~form~exists }
10839 {
10840     No~key~'\l_keys_key_str'~\l
10841     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10842 }
10843 \str_const:Ne \c_@@_available_keys_str
10844 {
10845     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10846     { For~a~list~of~the~available~keys,~type~H~<return>. }
10847 }
10848 \seq_new:N \g_@@_types_of_matrix_seq
10849 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10850 {
10851     NiceMatrix ,

```

```

10852     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10853 }
10854 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10855 { \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`.

```

10856 \cs_new_protected:Npn \@@_err_too_many_cols:
10857 {
10858   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10859   { \@@_fatal:nn { too-many-cols-for-array } }
10860   \int_compare:nNt \l_@@_last_col_int = { -2 }
10861   { \@@_fatal:n { too-many-cols-for-matrix } }
10862   \int_compare:nNt \l_@@_last_col_int = { -1 }
10863   { \@@_fatal:n { too-many-cols-for-matrix } }
10864   \bool_if:NF \l_@@_last_col_without_value_bool
10865   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10866 }

```

The following command must *not* be protected since it's used in an error message.

```

10867 \cs_new:Npn \@@_message_hdotsfor:
10868 {
10869   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
10870   { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
10871     \token_to_str:N \Hbrace \ is-incorrect. }
10872 }

```

```

10873 \cs_new_protected:Npn \@@_Hline_in_cell:
10874 { \@@_error:n { Misuse-of-Hline } }
10875 \@@_msg_new:nn { Misuse-of-Hline }
10876 {
10877   Misuse-of-\token_to_str:N \Hline. \\
10878   Error-in-the-row~ \int_use:N \c@iRow\ of-your-\@@_full_name_env:~
10879   \token_to_str:N \Hline\ (like-\token_to_str:N \hline)~must-be-used-only~
10880   at-the-beginning-of-a-row.~Maybe-you-have-forgotten-a-\token_to_str:N \\~
10881   Your-command-will-be-ignored.
10882 }

```

```

10883 \cs_new_protected:Npn \@@_hdashedline:
10884 { \@@_error:n { Misuse-of-hdashedline } }
10885 \cs_new_protected:Npn \@@_cdashedline:n #1
10886 { \@@_error:n { Misuse-of-cdashedline } }
10887 \@@_msg_new:nn { Misuse-of-hdashedline }
10888 {
10889   You-should-load-TikZ~(with-\token_to_str:N \usepackage \{tikz\})~in-order~
10890   to-be-able-to-use-the-command-\token_to_str:N \hdashedline.~Your-command~
10891   will-be-ignored.
10892 }
10893 \@@_msg_new:nn { Misuse-of-cdashedline }
10894 {
10895   You-should-load-TikZ~(with-\token_to_str:N \usepackage \{tikz\})~in-order~
10896   to-be-able-to-use-the-command-\token_to_str:N \cdashedline.~Your-command~
10897   will-be-ignored.
10898 }
10899 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10900 {
10901   Incompatible-options.\\
10902   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-the-same-time.~
10903   The-output-will-not-be-reliable.
10904 }

```

```

10905 \@@_msg_new:nn { Body~alone }
10906 {
10907   \token_to_str:N \Body\ alone. \\
10908   You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10909 }
10910 \@@_msg_new:nn { Misuse-of~CodeBefore }
10911 {
10912   Misuse-of~\token_to_str:N \CodeBefore. \\
10913   \token_to_str:N \CodeBefore\ must~be-used-only-at~the~beginning-of~
10914   the~environment.
10915 }
10916 \@@_msg_new:nn { NiceTabularX~probably~required }
10917 {
10918   Incorrect~syntax.\\
10919   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10920   ~width~that~you~wish~for~your~tabular.~But~you~can~also~use~\{NiceTabular\}~
10921   with~the~key~'width'.
10922 }
10923 \@@_msg_new:nn { cellcolor~in~Block }
10924 {
10925   Misuse-of~\token_to_str:N \cellcolor \\
10926   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10927   \bool_if:NTF \l_@@_amp_in_blocks_bool
10928     { (but~you~could~use~it~in~a~sub~block~since~'&-in-blocks'~is~in~force) }
10929     { (it's~possible~in~a~sub~block~when~'&-in-blocks'~is~in~force) }
10930   .~Here,~you~should~use~the~key~'fill'~of~the~block.~Your~command~will~be~ignored.
10931 }
10932 \@@_msg_new:nn { rowcolor~in~Block }
10933 {
10934   Misuse-of~\token_to_str:N \rowcolor \\
10935   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.~
10936   However,~it's~possible~to~color~the~block~with~its~key~'fill'.~
10937   Your~command~will~be~ignored.
10938 }
10939 \@@_msg_new:nn { key~color~inside }
10940 {
10941   Deleted~key.\\
10942   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10943   ~'nicematrix'~and~must~not~be~used.
10944 }
10945 \@@_msg_new:nn { Misuse-of~FalseRow }
10946 {
10947   Misuse-of~\token_to_str:N \FalseRow \\
10948   Error~in~the~row~ \int_use:N \c@iRow\ of~your~\@@_full_name_env:~
10949   \token_to_str:N \FalseRow\ must~be~used~only~at~the~beginning~of~a~row.~
10950   Your~command~will~be~ignored.
10951 }
10952 \@@_msg_new:nn { Invalid~argument~for~w }
10953 {
10954   Invalid~argument~for~w.\\
10955   You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10956   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10957   If~you~go~on,~'c'~will~be~used.
10958 }
10959 \@@_msg_new:nn { invalid~weight }
10960 {
10961   Unknown~key.\\
10962   The~key~'\l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.~
10963   The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10964   \IfPackageLoadedTF { varwidth }

```

```

10965     { (since~'varwidth'~is~loaded)~}
10966     { (if~you~load~'varwidth')~}
10967     and~real~numbers~for~the~weight~of~the~X~column.
10968 }

10969 \@@_msg_new:nn { last~col~not~used }
10970 {
10971     Column~not~used.\\
10972     The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
10973     in~your~\@@_full_name_env: .~However,~you~can~go~on.
10974 }

10975 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10976 {
10977     Too~many~columns.\\
10978     In~the~row~ \int_eval:n { \c@iRow },~
10979     you~try~to~use~more~columns~
10980     than~allowed~by~your~ \@@_full_name_env: .
10981     \@@_message_hdotsfor: \
10982     The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10983     (plus~the~exterior~columns).~
10984     Maybe~you~have~forgotten~a~\token_to_str:N \\
10985 }

10986 \@@_msg_new:nn { too~many~cols~for~matrix }
10987 {
10988     Too~many~columns.\\
10989     In~the~row~ \int_eval:n { \c@iRow },~
10990     you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10991     \@@_message_hdotsfor: \
10992     Recall~that~the~maximal~number~of~columns~for~a~matrix~
10993     (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10994     LaTeX~counter~'MaxMatrixCols'.~
10995     Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10996     (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10997     Maybe,~you~have~forgotten~a~\token_to_str:N \\
10998 }

10999 \cs_set:Npn \@@_message_about_false_cols:
11000 {
11001     \int_compare:nNnTF { \clist_count:N \g_@@_false_cols_clist } = 1
11002     { (including~one~false~column~F)~ }
11003     {
11004         \int_compare:nNnT { \clist_count:N \g_@@_false_cols_clist } > 1
11005         {
11006             (including~\int_to_arabic:n { \clist_count:N \g_@@_false_cols_clist }~
11007             false~columns~F)~
11008         }
11009     }
11010 }

11011 \@@_msg_new:nn { too~many~cols~for~array }
11012 {
11013     Too~many~columns.\\
11014     In~the~row~ \int_eval:n { \c@iRow },~
11015     ~you~try~to~use~more~columns~than~allowed~by~your~
11016     \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
11017     \int_use:N \g_@@_static_num_of_col_int \
11018     \@@_message_about_false_cols:
11019     \bool_if:nT
11020     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
11021     { (plus~the~exterior~ones)~}
11022     since~the~preamble~is~' \g_@@_user_preamble_tl '~
11023     Maybe,~you~have~forgotten~a~\token_to_str:N \\
11024 }

```

```

11025 \@@_msg_new:nn { columns-not-used }
11026 {
11027   Columns-not-used.\
11028   The-preamble-of-your~ \@@_full_name_env: \ is~
11029   '\exp_not:V \g_@@_user_preamble_tl'.~
11030   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
11031   columns~\@@_message_about_false_cols:
11032   but-you-only-used~ \int_use:N \c@jCol .~The-columns-that-you-did-not~
11033   used-won't-be-created.~You-won't-have-similar-warning-till-the-end-of-the-document.
11034 }

11035 \str_const:Nn \c_@@_explanation_no_cell_nodes_str
11036 {
11037   because-the-key~'no-cell-nodes'~is-in-force~
11038   (you-should-deactivate-the-key~'no-cell-nodes'~whose-only-goal~
11039   is-to-speed-compilation-up)
11040 }

11041 \@@_msg_new:nn { xdots-without-cell-nodes }
11042 {
11043   Command~#1 forbidden.\
11044   You-can't-use-the-command~#1~\c_@@_explanation_no_cell_nodes_str.~
11045   Your-command-will-be-ignored.
11046 }

11047 \@@_msg_new:nn { Misuse-of-NiceTabularNotes }
11048 {
11049   Misuse-of~\token_to_str:N \NiceTabularNotes \
11050   \token_to_str:N~\NiceTabularNotes\ should-be-used-only-after-at-tabular~
11051   which-uses~notes/no-print`.~ Your-command-will-be-ignored.
11052 }

11053 \@@_msg_new:nn { empty-preamble }
11054 {
11055   Empty-preamble.\
11056   The-preamble-of-your~ \@@_full_name_env: \ is-empty.
11057 }

11058 \@@_msg_new:nn { in-first-col }
11059 {
11060   Misuse-of~#1\
11061   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.~
11062   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-col'.
11063 }

11064 \@@_msg_new:nn { in-last-col }
11065 {
11066   Misuse-of~#1\
11067   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.~
11068   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-col'.
11069 }

11070 \@@_msg_new:nn { in-first-row }
11071 {
11072   Misuse-of~#1\
11073   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.~
11074   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-row'.
11075 }

11076 \@@_msg_new:nn { in-last-row }
11077 {
11078   Misuse-of~#1\
11079   You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.~
11080   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-row'.
11081 }

11082 \@@_msg_new:nn { TopRule-without-booktabs }
11083 {
11084   Misuse-of~#1\

```

```

11085     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
11086     You~should~load~'booktabs'~(before~or~after~'nicematrix').~
11087     Your~command~will~be~ignored.
11088 }

11089 \@@_msg_new:nn{ rotate~in~p~col }
11090 {
11091     \token_to_str:N \rotate\ forbidden.\\
11092     You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
11093     'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X' }.~
11094     If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
11095     consider~the~classical~command~\token_to_str:N \rotatebox.
11096 }

11097 \@@_msg_new:nn { Not~empty~cell~in~false~column }
11098 {
11099     Not~empty~cell.\\
11100     The~cell~( #1~#2 )~of~your~\@@_full_name_env: \
11101     is~not~empty~although~the~column~#2~is~a~
11102     "false~column"~(letter~F~in~the~preamble).~
11103     \bool_if:NTF \l_@@_light_syntax_bool
11104     { You~should~put~\{\}~for~an~empty~cell. }
11105     {
11106         \int_compare:nNt { #2 } > 1
11107         { Recall~that~you~have~to~put~*two*~ampersands~(&&). }
11108     }
11109 }

11110 \@@_msg_new:nn { TopRule~without~tikz }
11111 {
11112     Misuse~of~#1\\
11113     You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.~
11114     You~should~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\}).~
11115     \IfPackageLoadedF { booktabs }
11116     { You~should~also~load~'booktabs'~
11117       with~\token_to_str:N \usepackage \{booktabs\}.~ }
11118     Your~command~will~be~ignored.
11119 }

11120 \@@_msg_new:nn { caption~outside~float }
11121 {
11122     Key~caption~forbidden.\\
11123     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
11124     environment~(such~as~\{table\}).~Your~key~will~be~ignored.
11125 }

11126 \@@_msg_new:nn { short~caption~without~caption }
11127 {
11128     You~should~not~use~the~key~'short~caption'~without~'caption'.~
11129     However,~your~'short~caption'~will~be~used~as~'caption'.
11130 }

11131 \@@_msg_new:nn { double~closing~delimiter }
11132 {
11133     Double~delimiter.\\
11134     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
11135     delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
11136     \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
11137 }

11138 \@@_msg_new:nn { delimiter~after~opening }
11139 {
11140     Double~delimiter.\\
11141     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
11142     delimiter.~That~delimiter~will~be~ignored.
11143 }

11144 \@@_msg_new:nn { bad~option~for~line~style }

```

```

11145 {
11146   Bad-line-style.\
11147   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line-style'~
11148   is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
11149   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
11150 }
11151 \@@_msg_new:nn { draw-trees~with~no-cell-nodes }
11152 {
11153   Incompatible-keys.\
11154   You~can't~use~the~key~'draw-trees-in-col'~
11155   here~\c_@@_explanation_no_cell_nodes_str.~
11156   If~you~go~on,~that~key~will~be~ignored.
11157 }
11158 \@@_msg_new:nn { corners~with~no-cell-nodes }
11159 {
11160   Incompatible-keys.\
11161   You~can't~use~the~key~'corners'~here~\c_@@_explanation_no_cell_nodes_str.~
11162   If~you~go~on,~that~key~will~be~ignored.
11163 }
11164 \@@_msg_new:nn { extra-nodes~with~no-cell-nodes }
11165 {
11166   Incompatible-keys.\
11167   You~can't~create~'extra~nodes'~here~\c_@@_explanation_no_cell_nodes_str.~
11168   If~you~go~on,~those~extra~nodes~won't~be~created.
11169 }
11170 \@@_msg_new:nn { Identical~notes~in~caption }
11171 {
11172   Identical~tabular~notes.\
11173   You~can't~put~several~notes~with~the~same~content~in~
11174   \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
11175   If~you~go~on,~the~output~will~probably~be~erroneous.
11176 }
11177 \@@_msg_new:nn { tabularnote~below~the~tabular }
11178 {
11179   \token_to_str:N \tabularnote \ forbidden\
11180   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
11181   of~your~tabular~because~the~caption~will~be~composed~below~
11182   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
11183   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
11184   Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
11185   no~similar~error~will~raised~in~this~document.
11186 }
11187 \@@_msg_new:nn { Unknown~key~for~rules }
11188 {
11189   Unknown~key.\
11190   There~are~only~three~keys~available~here:~'width',~'color'~and~
11191   'fix-vertex'.~Your~key~' \l_keys_key_str '~will~be~ignored.
11192 }
11193 \@@_msg_new:nn { Unknown~key~for~trees }
11194 {
11195   Unknown~key.\
11196   There~are~only~three~keys~available~here:~width~color~and~
11197   rounded-corners.~Your~key~' \l_keys_key_str '~will~be~ignored.
11198 }
11199 % \end{macrocode}
11200 %
11201 %
11202 % \begin{macrocode}
11203 \@@_msg_new:nn { Unknown~key~for~Hbrace }
11204 {
11205   Unknown~key.\

```

```

11206     You-have-used-the-key~' \l_keys_key_str '~but-the-only~
11207     keys-allowed-for-the-commands~ \token_to_str:N \Hbrace \
11208     and~ \token_to_str:N \Vbrace \ are::~'brace-shift(+)',~'color',~
11209     'horizontal-label(s)',~'shorten'~'shorten-end'~
11210     and~'shorten-start'.
11211 }
11212 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
11213 {
11214     Unknown~key.\\
11215     There-is-only~two-keys-available-here::~
11216     'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will-be-ignored.
11217 }
11218 \@@_msg_new:nn { Unknown~key~for~rotate }
11219 {
11220     Unknown~key.\\
11221     The-only-keys-available-here-are~'c'~and~-90'.~
11222     Your~key~' \l_keys_key_str '~will-be-ignored.
11223 }
11224 \@@_msg_new:nnn { Unknown~key~for~custom-line }
11225 {
11226     Unknown~key.\\
11227     The-key~' \l_keys_key_str '~is-unknown-in-a~'custom-line'.~
11228     It~you-go-on,~you-will-probably-have-other-errors. \\
11229     \c_@@_available_keys_str
11230 }
11231 {
11232     The-available-keys-are~(in~alphabetic-order)::~
11233     ccommand,~
11234     color,~
11235     dashed \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11236     command,~
11237     dotted,~
11238     letter,~
11239     multiplicity,~
11240     sep-color,~
11241     tikz \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11242     and~total-width.
11243 }
11244 \@@_msg_new:nnn { Unknown~key~for~default-line }
11245 {
11246     Unknown~key.\\
11247     The-key~' \l_keys_key_str '~is-unknown-in-a~'default-line'.~
11248     It~you-go-on,~you-will-probably-have-other-errors. \\
11249     \c_@@_available_keys_str
11250 }
11251 {
11252     The-available-keys-are~(in~alphabetic-order)::~
11253     color,~
11254     dashed \IfPackageLoadedF { tikz } { ~ (when~TikZ~is~loaded)} ,~
11255     dotted,~
11256     multiplicity,~
11257     sep-color,~
11258     tikz~\IfPackageLoadedF { tikz } { (when~TikZ~is~loaded)~}
11259     and~total-width.
11260 }
11261 \@@_msg_new:nn { dashed~for~default-line~without~TikZ}
11262 {
11263     Key~'dashed'~forbidden.\\
11264     If~you-want~to~use~the~value~'dashed'~for~'default-line',~
11265     you-must-load-TikZ~(with~\token_to_str:N \usepackage \{tikz\}).
11266 }

```

```

11267 \@@_msg_new:nnn { Unknown~key~for~xdots }
11268 {
11269   Unknown~key.\\
11270   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
11271   \c_@@_available_keys_str
11272 }
11273 {
11274   The~available~keys~are~(in~alphabetic~order):~
11275   'color',~
11276   'horizontal(s)-labels',~
11277   'inter',~
11278   'line-style',~
11279   'nullify',~
11280   'radius',~
11281   'shorten',~
11282   'shorten-end'~and~'shorten-start'.
11283 }
11284 \@@_msg_new:nn { Unknown~key~for~rowcolors }
11285 {
11286   Unknown~key.\\
11287   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
11288   (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
11289 }
11290 \@@_msg_new:nn { Col~outside~tabular~in~trees }
11291 {
11292   Error~with~'draw-trees-in-col' \\
11293   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
11294   is~\int_use:N \c@jCol.~It~will~be~ignored.
11295 }
11296 \@@_msg_new:nn { Last~col~in~trees }
11297 {
11298   Error~with~'draw-trees-in-col' \\
11299   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
11300   the~last~column~of~your~tabular.~It~will~be~ignored.
11301 }
11302 \@@_msg_new:nn { label~without~caption }
11303 {
11304   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
11305   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
11306 }
11307 \@@_msg_new:nn { W~warning }
11308 {
11309   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
11310   (row~ \int_use:N \c@iRow ).
11311 }
11312 \@@_msg_new:nn { Construct~too~large }
11313 {
11314   Construct~too~large.\\
11315   Your~command~ \token_to_str:N #1
11316   can't~be~drawn~because~your~matrix~is~too~small.~
11317   Your~command~will~be~ignored.
11318 }
11319 \@@_msg_new:nn { underscore~after~nicematrix }
11320 {
11321   Problem~with~'underscore'.\\
11322   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
11323   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
11324   ' \token_to_str:N \Cdots \token_to_str:N _
11325   \{ n \token_to_str:N \text \{ ~times \} \}'.
11326 }

```

```

11327 \@@_msg_new:nn { ampersand-in-light-syntax }
11328 {
11329   Ampersand~forbidden.\
11330   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
11331   ~the~key~'light-syntax'~is~in~force.
11332 }
11333 \@@_msg_new:nn { double-backslash-in-light-syntax }
11334 {
11335   Double~backslash~forbidden.\
11336   You~can't~use~ \token_to_str:N \
11337   ~to~separate~rows~because~the~key~'light-syntax'~
11338   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
11339   (set~by~the~key~'end-of-row').
11340 }
11341 \@@_msg_new:nn { bad-value-for-baseline }
11342 {
11343   Bad~value~for~baseline.\
11344   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11345   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
11346   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
11347   the~form~'line-i'.~A~value~of~1~will~be~used.
11348 }
11349 \@@_msg_new:nn { bad-value-for-baseline-line }
11350 {
11351   Bad~value~for~baseline~with~line.\
11352   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
11353   valid.~The~number~of~the~line~must~be~between~1~and~
11354   \int_eval:n { \c@iRow + 1 }.~
11355   A~value~of~'line-1'~will~be~used.
11356 }
11357 \@@_msg_new:nn { detection-of-empty-cells }
11358 {
11359   Problem~with~'not-empty'\
11360   For~technical~reasons,~you~must~activate~
11361   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
11362   in~order~to~use~the~key~' \l_keys_key_str '~
11363   Your~key~will~be~ignored.
11364 }
11365 \@@_msg_new:nn { siunitx-too-old }
11366 {
11367   siunitx~too~old\
11368   You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
11369   is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26)~and~
11370   the~version~that~you~have~loaded~has~the~date:~
11371   \str_range:cnn { ver @ siunitx . sty } { 1 } { 10 }.
11372 }
11373 \@@_msg_new:nn { Invalid-name }
11374 {
11375   Invalid~name.\
11376   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
11377   \SubMatrix \ of~your~ \@@_full_name_env: .~
11378   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
11379   Your~key~will~be~ignored.
11380 }
11381 \@@_msg_new:nn { Hbrace-not-allowed }
11382 {
11383   Command~not~allowed.\
11384   You~can't~use~the~command~ \token_to_str:N #1
11385   because~you~have~not~loaded~
11386   \IfPackageLoadedTF { tikz }
11387   { the~TikZ~library~'decorations.pathreplacing'.~Use~ }

```

```

11388     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
11389     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
11390     Your~command~will~be~ignored.
11391 }
11392 \@@_msg_new:nn { Vbrace~not~allowed }
11393 {
11394     Command~not~allowed.\\
11395     You~can't~use~the~command~ \token_to_str:N \Vbrace \
11396     because~you~have~not~loaded~TikZ~
11397     and~the~TikZ~library~'decorations.pathreplacing'.~
11398     Use: ~\token_to_str:N \usepackage \{tikz\}~
11399     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
11400     Your~command~will~be~ignored.
11401 }
11402 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
11403 {
11404     Wrong~line.\\
11405     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
11406     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
11407     number~is~not~valid.~It~will~be~ignored.
11408 }
11409 \@@_msg_new:nn { Impossible~SubMatrix }
11410 {
11411     Impossible~SubMatrix.\\
11412     It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
11413     because~all~the~cells~are~empty~in~the~column~on~the~#1~
11414     side~of~that~\token_to_str:N \SubMatrix.
11415     \bool_if:NT \l_@@_submatrix_slim_bool
11416     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
11417     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11418 }
11419 \@@_msg_new:nn { Impossible~SubMatrix~no~cell~nodes }
11420 {
11421     Impossible~SubMatrix.\\
11422     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
11423     \c_@@_explanation_no_cell_nodes_str.~
11424     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
11425 }
11426 \@@_msg_new:nnn { width~without~X~columns }
11427 {
11428     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
11429     the~preamble~(' \exp_not:V \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
11430     Your~key~will~be~ignored.
11431 }
11432 {
11433     This~message~is~the~message~'width~without~X~columns'~
11434     of~the~module~'nicematrix'.~
11435     The~experimented~users~can~disable~that~message~with~
11436     \token_to_str:N \msg_redirect_name:nnn .\\
11437 }
11438
11439 \@@_msg_new:nn { key~multiplicity~with~dotted }
11440 {
11441     Incompatible~keys. \\
11442     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
11443     in~a~'custom~line'.~They~are~incompatible.~
11444     The~key~'multiplicity'~will~be~ignored.
11445 }
11446 \@@_msg_new:nn { empty~environment }
11447 {
11448     Empty~environment.\\

```

```

11449     Your~ \@@_full_name_env: \ is~empty.
11450 }

11451 \@@_msg_new:nn { No~letter~and~no~command }
11452 {
11453     Erroneous~use.\\
11454     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
11455     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
11456     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
11457 }

11458 \@@_msg_new:nn { Forbidden~letter }
11459 {
11460     Forbidden~letter.\\
11461     You~can't~use~the~letter~'#1'~for~a~customized~line.~
11462     It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
11463 }

11464 \@@_msg_new:nn { Several~letters }
11465 {
11466     Wrong~name.\\
11467     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
11468     have~used~' \l_@@_letter_str ').~It~will~be~ignored.
11469 }

11470 \@@_msg_new:nn { Delimiter~with~small }
11471 {
11472     Delimiter~forbidden.\\
11473     You~can't~put~a~delimiter~in~the~preamble~of~your~
11474     \@@_full_name_env: \
11475     because~the~key~'small'~is~in~force.
11476 }

11477 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
11478 {
11479     Unknown~cell.\\
11480     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
11481     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
11482     can't~be~executed~because~a~cell~doesn't~exist.~
11483     Your~command~ \token_to_str:N \line \ will~be~ignored.
11484 }

11485 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
11486 {
11487     Duplicate~name. \\
11488     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
11489     in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
11490     \bool_if:NF \g_@@_messages_for_Overleaf_bool
11491     { For~a~list~of~the~names~already~used,~type~H~<return>. }
11492 }
11493 {
11494     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
11495     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
11496 }

11497 \@@_msg_new:nn { r~or~l~with~preamble }
11498 {
11499     Erroneous~use. \\
11500     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
11501     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
11502     your~ \@@_full_name_env: .~
11503     Your~key~will~be~ignored.
11504 }

11505 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11506 {
11507     Erroneous~use. \\
11508     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\

```

```

11509     in~an~exterior~column~of~the~array.
11510 }
11511 \@@_msg_new:nn { bad~corner }
11512 {
11513     Bad~corner. \\
11514     '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11515     'corners').~The~available~values~are:~NW,~SW,~NE,~SE,~N,~S,~W~and~E~
11516     This~specification~of~corner~will~be~ignored.
11517 }
11518 \@@_msg_new:nn { bad~border }
11519 {
11520     Bad~border. \\
11521     '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11522     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11523     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11524     also~use~the~key~'tikz'
11525     \IfPackageLoadedF { tikz }
11526     { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11527     This~specification~of~border~will~be~ignored.
11528 }
11529 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11530 {
11531     TikZ~not~loaded. \\
11532     You~can't~use~ \token_to_str:N \TikzEveryCell \
11533     because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11534     \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11535     Your~command~will~be~ignored.
11536 }
11537 \@@_msg_new:nn { tikz~key~without~tikz }
11538 {
11539     TikZ~not~loaded. \\
11540     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
11541     \Block '~because~you~have~not~loaded~TikZ.~
11542     You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11543     before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11544 }
11545 \@@_msg_new:nn { Bad~argument~for~Block }
11546 {
11547     Bad~argument. \\
11548     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11549     be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11550     '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
11551     the~argument~was~empty).
11552 }
11553 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11554 {
11555     Erroneous~use. \\
11556     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11557     'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11558     (the~value~' \l_keys_value_tl '~will~be~ignored).
11559 }
11560 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11561 {
11562     Erroneous~use. \\
11563     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11564     'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11565     (the~value~' \l_keys_value_tl '~will~be~ignored).
11566 }
11567 \@@_msg_new:nn { Block~too~large~1 }
11568 {

```

```

11569     Block~too~large. \\
11570     You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
11571     too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11572 }

11573 \@@_msg_new:nn { Block~too~large~2 }
11574 {
11575     Block~too~large. \\
11576     The~preamble~of~your~ \@@_full_name_env:,~
11577     which~is~ '\exp_not:V\g_@@_user_preamble_tl',~announces~
11578     \int_use:N \g_@@_static_num_of_col_int \ columns~
11579     \@@_message_about_false_cols:
11580     but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
11581     specified~in~the~cell~#1~#2~can't~be~drawn.~
11582     \bool_if:NF \l_@@_light_syntax_bool
11583     {
11584         You~should~add~some~ampersands~(&)~at~the~end~of~the~first~row~
11585         of~your~ \@@_full_name_env:~.~
11586     }
11587     This~block~will~be~ignored.
11588 }

11589 \@@_msg_new:nn { unknown~column~type }
11590 {
11591     Bad~column~type. \\
11592     The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.
11593 }

11594 \@@_msg_new:nn { unknown~column~type~multicolumn }
11595 {
11596     Bad~column~type. \\
11597     The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
11598     ~of~your~ \@@_full_name_env: \ is~unknown.
11599 }

11600 \@@_msg_new:nn { unknown~column~type~S }
11601 {
11602     Bad~column~type. \\
11603     The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown.~
11604     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
11605     load~that~package~with~\token_to_str:N \usepackage \{siunitx\}.
11606 }

11607 \@@_msg_new:nn { unknown~column~type~S~multicolumn }
11608 {
11609     Bad~column~type. \\
11610     The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
11611     of~your~ \@@_full_name_env: \ is~unknown.~If~you~want~to~use~the~column~
11612     type~'S'~of~siunitx,~you~should~load~that~package~(with~
11613     \token_to_str:N \usepackage \{siunitx\}).
11614 }

11615 \@@_msg_new:nn { unknown~column~type~; }
11616 {
11617     Bad~column~type. \\
11618     The~column~type~';'~in~your~ \@@_full_name_env: \ is~unknown.~
11619     You~should~load~TikZ~(with~\token_to_str:N \usepackage \{tikz\})~
11620     in~order~to~use~it~for~vertical~dashed~rules.
11621 }

11622 \@@_msg_new:nn { unknown~column~type~RCL }
11623 {
11624     Bad~column~type. \\
11625     The~column~type~'#1'~in~your~ \@@_full_name_env: \ is~unknown.~
11626     Maybe~you~want~to~use~\str_lowercase:n { #1 }.
11627 }

11628 \@@_msg_new:nn { tabularnote~forbidden }

```

```

11629 {
11630   Forbidden-command. \\
11631   You-can't-use-the-command~ \token_to_str:N \tabularnote \
11632   ~here.~This-command-is-available-only-in~
11633   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
11634   the-argument-of-a-command~\token_to_str:N \caption \ included~
11635   in-an-environment~\{table\}.~Your-command-will-be-ignored.
11636 }
11637 \@@_msg_new:nn { borders~forbidden }
11638 {
11639   Forbidden-key.\\
11640   You-can't-use-the-key~'borders'~of~the-command~ \token_to_str:N \Block \
11641   because-the-option~'rounded-corners'~is-in-force-with-a-non-zero-value.~
11642   Your-key-will-be-ignored.
11643 }
11644 \@@_msg_new:nn { bottomrule~without~booktabs }
11645 {
11646   booktabs-not-loaded.\\
11647   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
11648   loaded~'booktabs'.~You-should-load~'booktabs',~before-or~
11649   after~'nicematrix'.~Your-key-will-be-ignored.
11650 }
11651 \@@_msg_new:nn { enumitem~not~loaded }
11652 {
11653   enumitem-not-loaded. \\
11654   You-can't-use-the-command~ \token_to_str:N \tabularnote \
11655   ~because-you-haven't~loaded~'enumitem'.~We-should-load-it~
11656   (before-or-after~'nicematrix').~All-the-commands~
11657   \token_to_str:N \tabularnote \ will-be-ignored-in-the-document.
11658 }
11659 \@@_msg_new:nn { tikz~without~tikz }
11660 {
11661   TikZ-not-loaded. \\
11662   You-can't-use-the-key~'tikz'~here~because-TikZ-is-not-loaded.~You~
11663   should-load-TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11664   If-you-go-on,~your-key-will-be-ignored.
11665 }
11666 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11667 {
11668   TikZ-not-loaded. \\
11669   You-have-used-the-key~'tikz'~(or~'dashed')~in-the-definition-of-a~
11670   customized~line~(with~'custom~line')~but~TikZ-is-not-loaded.~
11671   You-can-go-on-but-you-will-have-another~error~if-you-actually~
11672   use-that-custom~line.~You-should-load-TikZ~(with~
11673   \token_to_str:N \usepackage \{tikz\}).
11674 }
11675 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11676 {
11677   TikZ-not-loaded. \\
11678   You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
11679   command~' \token_to_str:N \Block ')~but~TikZ-is-not-loaded.~
11680   Your-key-will-be-ignored.~You-should-load-TikZ~(with~
11681   \token_to_str:N \usepackage \{tikz\}).
11682 }
11683 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11684 {
11685   Erroneous-use.\\
11686   In-a~'custom~line',~you-have-used-both~'tikz'~(or~'dashed')~and~'color',~
11687   which-is-forbidden~(you-should-use~'color'~inside~the-key~'tikz'~itself).~
11688   The-key~'color'~will-be-ignored.
11689 }

```

```

11690 \@@_msg_new:nn { Wrong-last-row }
11691 {
11692   Wrong-number.\
11693   You-have-used-'last-row'= \int_use:N \l_@@_last_row_int '~but-your~
11694   \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
11695   If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
11696   last-row-but-you-should-correct-your-code.~You-can-avoid-this~
11697   problem-by-using-'last-row'~without-value~(more-compilations~
11698   might-be-necessary).
11699 }
11700 \@@_msg_new:nn { Yet-in-env }
11701 {
11702   Nested-environments.\
11703   Environments-of~nicematrix~can't-be-nested.~However-you~
11704   can-insert,~for-example,~a~\{tabular\}~in-a~\{NiceTabular\}~
11705   or~a~\{NiceTabular\}~in-a~\{tabular\}.~You-can-also-compose~
11706   an-environment-of~nicematrix~in-a-box-of~LaTeX~and-insert~
11707   that~box~in~another-environment-of~nicematrix.
11708 }
11709 \@@_msg_new:nn { Outside-math-mode }
11710 {
11711   Outside-math-mode.\
11712   The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
11713   (and-not-in~ \token_to_str:N \vcenter ).
11714 }
11715 \@@_msg_new:nn { One-letter-allowed }
11716 {
11717   Bad-name.\
11718   The-value-of-key~' \l_keys_key_str '~must-be-of-length-1~and~
11719   you-have-used~' \l_keys_value_tl '~Your-key-will-be-ignored.
11720 }
11721 \@@_msg_new:nn { TabularNote-in-CodeAfter }
11722 {
11723   Environment~\{TabularNote\}~forbidden.\
11724   You-must-use~\{TabularNote\}~at-the-end-of-your~\{NiceTabular\}~
11725   but~*before*~the~ \token_to_str:N \CodeAfter .~Your-environment~
11726   \{TabularNote\}~will-be-ignored.
11727 }
11728 \@@_msg_new:nn { varwidth-not-loaded }
11729 {
11730   varwidth-not-loaded.\
11731   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
11732   loaded.~You-should-load~'varwidth',~before-or-after~'nicematrix'.~
11733   Your-column-will-behave-like~'p'.
11734 }
11735 \@@_msg_new:nn { varwidth-not-loaded-in-X }
11736 {
11737   varwidth-not-loaded.\
11738   You-can't-use-the-key~'V'~in-your-column~'X'~
11739   because~'varwidth'~is-not-loaded.~You-should-load~'varwidth',~
11740   before-or-after~'nicematrix'.~ Your-key-will-be-ignored.
11741 }
11742 \@@_msg_new:nnn { Unknown-key-for-a-rule }
11743 {
11744   Unknown-key.\
11745   Your-key~' \l_keys_key_str '~is-unknown-for-a-rule.\
11746   \c_@@_available_keys_str
11747 }
11748 {
11749   The-available-keys-are:~color,~multiplicity,~sep-color,~tikz~
11750   and-total-width~(the-latter-is-meaningful-only-in-cunjunction-with-'tikz').

```

11751 }

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `..`.

```

11752 \@@_msg_new:nnn { Unknown~key~for~Block }
11753 {
11754   Unknown~key. \\
11755   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11756   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11757   \c_@@_available_keys_str
11758 }
11759 {
11760   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11761   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11762   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/color,~rules/width,
11763   ~t,~T,~tikz,~transparent~and~vlines.
11764 }
11765 \@@_msg_new:nnn { Unknown~key~for~Brace }
11766 {
11767   Unknown~key. \\
11768   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11769   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11770   Your~key~will~be~ignored. \\
11771   \c_@@_available_keys_str
11772 }
11773 {
11774   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11775   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11776   right~shorten)~and~yshift.
11777 }
11778 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11779 {
11780   Unknown~key. \\
11781   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11782   \c_@@_available_keys_str
11783 }
11784 {
11785   The~available~keys~are~(in~alphabetic~order):~
11786   delimiters/color,~
11787   rules~(with~the~subkeys~'color'~and~'width'),~
11788   sub~matrix~(several~subkeys)~
11789   and~xdots~(several~subkeys).~
11790   The~latter~is~for~the~command~ \token_to_str:N \line .
11791 }
11792 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11793 {
11794   Unknown~key. \\
11795   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11796   \c_@@_available_keys_str
11797 }
11798 {
11799   The~available~keys~are~(in~alphabetic~order):~
11800   create~cell~nodes,~
11801   delimiters/color~and~
11802   sub~matrix~(several~subkeys).
11803 }
11804 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11805 {
11806   Unknown~key. \\
11807   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11808   \c_@@_available_keys_str

```

```

11809 }
11810 {
11811     The-available-keys-are~(in~alphabetic-order):~
11812     'delimiters/color',~
11813     'extra-height',~
11814     'hlines',~
11815     'hvlines',~
11816     'left-xshift',~
11817     'name',~
11818     'right-xshift',~
11819     'rules'~(with~the~subkeys~'color'~and~'width'),~
11820     'slim',~
11821     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11822 }
11823 \@@_msg_new:nnn { Unknown~key~for~notes }
11824 {
11825     Unknown~key.\\
11826     The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11827     \c_@@_available_keys_str
11828 }
11829 {
11830     The-available-keys-are~(in~alphabetic-order):~
11831     bottomrule,~
11832     code-after,~
11833     code-before(+),~
11834     detect-duplicates,~
11835     enumitem-keys,~
11836     enumitem-keys-para,~
11837     para,~
11838     label-in-list,~
11839     label-in-tabular~and~
11840     style.
11841 }
11842 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11843 {
11844     Unknown~key.\\
11845     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11846     \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11847     \c_@@_available_keys_str
11848 }
11849 {
11850     The-available-keys-are~(in~alphabetic-order):~
11851     bold,~
11852     cell-space-top-limit(+),~
11853     cell-space-bottom-limit(+),~
11854     cell-space-limits(+),~
11855     color,~
11856     fill~(alias:~rowcolor),~
11857     nb-rows,~
11858     opacity~and~
11859     rounded-corners.
11860 }
11861 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11862 {
11863     Unknown~key.\\
11864     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11865     \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11866     \c_@@_available_keys_str
11867 }
11868 {
11869     The-available-keys-are~(in~alphabetic-order):~
11870     &-in-blocks,~
11871     allow-duplicate-names,~

```

```

11872    ampersand-in-blocks,~
11873    caption-above,~
11874    cell-space-bottom-limit(+),~
11875    cell-space-limits(+),~
11876    cell-space-top-limit(+),~
11877    code-for-first-col(+),~
11878    code-for-first-row(+),~
11879    code-for-last-col(+),~
11880    code-for-last-row(+),~
11881    corners,~
11882    custom-key,~
11883    create-extra-nodes,~
11884    create-medium-nodes,~
11885    create-large-nodes,~
11886    custom-line,~
11887    delimiters~(several~subkeys),~
11888    end-of-row,~
11889    first-col,~
11890    first-row,~
11891    h(v)lines,~
11892    h(v)lines-except-borders,~
11893    last-col,~
11894    last-row,~
11895    left-margin,~
11896    light-syntax,~
11897    light-syntax-expanded,~
11898    matrix/columns-type,~
11899    no-cell-nodes,~
11900    notes~(several~subkeys),~
11901    nullify-dots,~
11902    pgf-node-code,~
11903    renew-dots,~
11904    renew-matrix,~
11905    respect-arraystretch,~
11906    rounded-corners,~
11907    right-margin,~
11908    rules~(with~the~subkeys~'color'~and~'width'),~
11909    small,~
11910    sub-matrix~(several~subkeys),~
11911    vlines,~
11912    width-of-false(+),~
11913    xdots~(several~subkeys).
11914 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

11915 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11916 {
11917   Unknown~key.\\
11918   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11919   \{NiceArray\}.~Your~key~will~be~ignored. \\
11920   \c_@@_available_keys_str
11921 }
11922 {
11923   The~available~keys~are~(in~alphabetic~order):~
11924   &-in-blocks,~
11925   ampersand-in-blocks,~
11926   b,~
11927   baseline,~
11928   c,~
11929   cell-space-bottom-limit,~
11930   cell-space-limits,~
11931   cell-space-top-limit,~
11932   code-after,~

```

```

11933 code-for-first-col(+),~
11934 code-for-first-row(+),~
11935 code-for-last-col(+),~
11936 code-for-last-row(+),~
11937 columns-width,~
11938 corners,~
11939 create-blocks-in-col,~
11940 create-extra-nodes,~
11941 create-medium-nodes,~
11942 create-large-nodes,~
11943 draw-trees-in-col,~
11944 extra-left-margin,~
11945 extra-right-margin,~
11946 first-col,~
11947 first-row,~
11948 h(v)lines,~
11949 h(v)lines-except-borders,~
11950 last-col,~
11951 last-row,~
11952 left-margin,~
11953 light-syntax,~
11954 light-syntax-expanded,~
11955 name,~
11956 no-cell-nodes,~
11957 nullify-dots,~
11958 pgf-node-code,~
11959 renew-dots,~
11960 respect-arraystretch,~
11961 right-margin,~
11962 rounded-corners,~
11963 rules~(with~the~subkeys~'color'~and~'width'),~
11964 small,~
11965 t,~
11966 vlines,~
11967 width-of-false(+),~
11968 xdots/color,~
11969 xdots/shorten-start(+),~
11970 xdots/shorten-end(+),~
11971 xdots/shorten(+)-and~
11972 xdots/line-style.
11973 }

```

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`).

```

11974 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11975 {
11976   Unknown~key.\\
11977   The~key~' \l_keys_key_str '~is~unknown~for~the~
11978   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11979   \c_@@_available_keys_str
11980 }
11981 {
11982   The~available~keys~are~(in~alphabetic~order):~
11983   &-in-blocks,~
11984   ampersand-in-blocks,~
11985   b,~
11986   baseline,~
11987   c,~
11988   cell-space-bottom-limit,~
11989   cell-space-limits,~
11990   cell-space-top-limit,~
11991   code-after,~
11992   code-for-first-col(+),~

```

```

11993 code-for-first-row(+),~
11994 code-for-last-col(+),~
11995 code-for-last-row(+),~
11996 columns-type,~
11997 columns-width,~
11998 corners,~
11999 create-blocks-in-col,~
12000 create-extra-nodes,~
12001 create-medium-nodes,~
12002 create-large-nodes,~
12003 draw-trees-in-col,~
12004 extra-left-margin,~
12005 extra-right-margin,~
12006 first-col,~
12007 first-row,~
12008 h(v)lines,~
12009 h(v)lines-except-borders,~
12010 l,~
12011 last-col,~
12012 last-row,~
12013 left-margin,~
12014 light-syntax,~
12015 light-syntax-expanded,~
12016 name,~
12017 no-cell-nodes,~
12018 nullify-dots,~
12019 pgf-node-code,~
12020 r,~
12021 renew-dots,~
12022 respect-arraystretch,~
12023 right-margin,~
12024 rounded-corners,~
12025 rules~(with~the~subkeys~'color'~and~'width'),~
12026 small,~
12027 t,~
12028 vlines,~
12029 width-of-false(+),~
12030 xdots/color,~
12031 xdots/shorten-start(+),~
12032 xdots/shorten-end(+),~
12033 xdots/shorten(+)-and~
12034 xdots/line-style.
12035 }

12036 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
12037 {
12038   Unknown~key.\\
12039   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
12040   \{NiceTabular\}.~Your~key~will~be~ignored. \\
12041   \c_@@_available_keys_str
12042 }
12043 {
12044   The~available~keys~are~(in~alphabetic~order):~
12045   &-in-blocks,~
12046   ampersand-in-blocks,~
12047   b,~
12048   baseline,~
12049   c,~
12050   caption,~
12051   cell-space-bottom-limit,~
12052   cell-space-limits,~
12053   cell-space-top-limit,~
12054   code-after,~
12055   code-for-first-col(+),~

```

```

12056 code-for-first-row(+),~
12057 code-for-last-col(+),~
12058 code-for-last-row(+),~
12059 columns-width,~
12060 corners,~
12061 custom-line,~
12062 create-blocks-in-col,~
12063 create-extra-nodes,~
12064 create-medium-nodes,~
12065 create-large-nodes,~
12066 draw-trees-in-col,~
12067 extra-left-margin,~
12068 extra-right-margin,~
12069 first-col,~
12070 first-row,~
12071 h(v)lines,~
12072 h(v)lines-except-borders,~
12073 label,~
12074 last-col,~
12075 last-row,~
12076 left-margin,~
12077 light-syntax,~
12078 light-syntax-expanded,~
12079 name,~
12080 no-cell-nodes,~
12081 notes~(several~subkeys),~
12082 nullify-dots,~
12083 pgf-node-code,~
12084 renew-dots,~
12085 respect-arraystretch,~
12086 right-margin,~
12087 rounded-corners,~
12088 rules~(with~the~subkeys~'color'~and~'width'),~
12089 short-caption,~
12090 t,~
12091 tabularnote,~
12092 vlines,~
12093 width-of-false(+),~
12094 xdots/color,~
12095 xdots/shorten-start(+),~
12096 xdots/shorten-end(+),~
12097 xdots/shorten(+)-and~
12098 xdots/line-style.
12099 }

12100 \@@_msg_new:nnn { Duplicate-name }
12101 {
12102   Duplicate-name.\
12103   The-name~' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
12104   the-same-environment-name-twice.~You-can-go-on,~but,~
12105   maybe,~you-will-have-incorrect-results-especially~
12106   if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
12107   message-again,~use-the-key~'allow-duplicate-names'~in~
12108   ' \token_to_str:N \NiceMatrixOptions '.\
12109   \bool_if:NF \g_@@_messages_for_Overleaf_bool
12110     { For-a-list-of-the-names-already-used,~type-H<return>. }
12111 }
12112 {
12113   The-names-already-defined-in-this-document-are:~
12114   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
12115 }

12116 \@@_msg_new:nn { caption-above~in~env }
12117 {
12118   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.~

```

```

12119     Your~key~will~be~ignored.
12120 }
12121 \@@_msg_new:nn { show-cell-names }
12122 {
12123     There~is~no~key~'show-cell-names'~in~nicematrix.\\
12124     You~should~use~the~command~\token_to_str:N \ShowCellNames\
12125     in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
12126     \CodeAfter.~Your~key~will~be~ignored.
12127 }
12128 \@@_msg_new:nn { Option~auto~for~columns~width }
12129 {
12130     Erroneous~use.\\
12131     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
12132     Your~key~will~be~ignored.
12133 }
12134 \@@_msg_new:nn { NiceTabularX~without~X }
12135 {
12136     NiceTabularX~without~X.\\
12137     You~should~not~use~\{NiceTabularX\}~without~X~columns.~However,~you~can~go~on.
12138 }
12139 \@@_msg_new:nn { Preamble~forgotten }
12140 {
12141     Preamble~forgotten.\\
12142     You~have~probably~forgotten~the~preamble~of~your~
12143     \@@_full_name_env: .
12144 }
12145 \@@_msg_new:nn { Invalid~col~number }
12146 {
12147     Invalid~column~number.\\
12148     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
12149     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
12150     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12151 }
12152 \@@_msg_new:nn { Invalid~row~number }
12153 {
12154     Invalid~row~number.\\
12155     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
12156     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
12157     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
12158 }
12159 \@@_define_com:NNN p ( )
12160 \@@_define_com:NNN b [ ]
12161 \@@_define_com:NNN v | |
12162 \@@_define_com:NNN V \l \l
12163 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	38
9	The <code>\CodeBefore</code>	54
10	The environment <code>{NiceArrayWithDelims}</code>	58
11	Construction of the preamble of the array	63
12	The redefinition of <code>\multicolumn</code>	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>	220

30	The keyword <code>\CodeAfter</code>	222
31	The delimiters in the preamble	222
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>	236
35	The command <code>TikzEveryCell</code>	239
36	The key <code>draw-trees-in-col</code>	240
37	The key <code>create-blocks-in-col</code>	242
38	The command <code>\ShowCellNames</code>	243
39	We process the options at package loading	245
40	About the package underscore	246
41	Compatibility with <code>threeparttable</code>	246
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	247