

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

March 29, 2026

## Abstract

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

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}

8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your~LaTeX~release~is~too~old. \\
11    You~need~at~least~the~version~of~2025-06-01. \\
12    If~you~use~Overleaf,~you~need~at~least~"TeXLive~2025".\\
13    The~package~'nicematrix'~won't~be~loaded.
14   }

15 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
16 \IfFormatAtLeastTF
17   { 2025-06-01 }
18   { }
19   { \msg_critical:nn { nicematrix } { latex-too-old } }
```

---

\*This document corresponds to the version 7.8a of `nicematrix`, at the date of 2026/03/29.

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

```

20 \RequirePackage { amsmath }

21 \RequirePackage{array}[=2025/06/08] % v2.6j

22 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
23 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
24 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
25 \cs_generate_variant:Nn \@@_error:nn { n e }
26 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
27 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
28 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
29 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }

```

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

```

30 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
31 {
32   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
33     { \msg_new:nnn { nicematrix } { #1 } { #2 \\\ #3 } }
34     {
35       \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).

```

36       \msg_new:nnn { nicematrix } { #1~+ } { #3 }
37     }
38   }

```

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

```

39 \cs_new_protected:Npn \@@_error_or_warning:n
40 {
41   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
42     \@@_warning:n
43     \@@_error:n
44 }

```

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”.

```

45 \bool_new:N \g_@@_messages_for_Overleaf_bool
46 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
47 {
48   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
49   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
50 }

51 \@@_msg_new:nn { mdwtab-loaded }
52 {
53   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
54   This~error~is~fatal.
55 }

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

```

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

```

58 \IfClassLoadedTF { revtex4-1 }
59 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
60 {
61   \IfClassLoadedTF { revtex4-2 }
62   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
63   {

```

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

```

64     \cs_if_exist:NT \rvtx@ifformat@geq
65     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
66     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
67   }
68 }
69 \@@_msg_new:nn { class~revtex }
70 {
71   You~can't~use~this~version~of~'nicematrix'~in~a~class~of~REVTeX~because~
72   REVTeX~is~*not*~compatible~with~recent~versions~of~LaTeX.~Sorry...\\
73   The~package~'nicematrix'~won't~be~loaded.
74 }
75 \bool_if:NT \c_@@_revtex_bool
76 { \msg_critical:nn { nicematrix } { class~revtex } }

```

## 2 Collecting options

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

*Example :*

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

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,  
 the command `\G` takes in an arbitrary number of optional arguments between square brackets.  
 Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

77 \cs_new_protected:Npn \@@_collect_options:n #1
78 {
79   \peek_meaning:NTF [
80     { \@@_collect_options:nw { #1 } }
81     { #1 { } }
82   }

```

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

```

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

```

### 3 Technical definitions

Here are definitions that have been added to the LaTeX kernel in February 2006.  
The following constants are defined only for efficiency in the tests.

```

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

103 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
104 \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).

```

105 \tl_new:N \l_@@_argspec_tl

106 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
107 \cs_generate_variant:Nn \str_set:Nn { N o }
108 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
109 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
110 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
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   {
181     \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
182     \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
183     \cs_new_protected:Npn \@@_revert_colortbl:
184     {
185       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
186       {
187         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
188         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
189       }
190     }

```

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).

```

191     \cs_new_protected:Npn \@@_replace_columncolor:
192     {
193       \tl_replace_all:Nnn \g_@@_array_preamble_tl
194       \columncolor
195       \@@_columncolor_preamble

```

`\@@_column_preamble`, despite its name, will be defined with `\NewDocumentCommand` because it takes in an optional argument between square brackets in first position for the colorimetric space.

```

196     }
197   }
198   {
199     \cs_new_protected:Npn \@@_revert_colortbl: { }
200     \cs_new_protected:Npn \@@_replace_columncolor:
201     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded.

```

202     \def \CT@arc@ { }
203     \def \arrayrulecolor #1 # { \CT@arc { #1 } }
204     \def \CT@arc #1 #2
205     {
206       \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
207       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
208     }

```

Idem for `\CT@drs@`.

```

209     \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
210     \def \CT@drs #1 #2
211     {
212       \dim_compare:nNnT \baselineskip = \c_zero_dim { \noalign }
213       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
214     }
215     \def \hline
216     {
217       \noalign { \ifnum 0 = ` } \fi
218       \cs_set_eq:NN \hskip \vskip
219       \cs_set_eq:NN \vrule \hrule
220       \cs_set_eq:NN \@width \@height
221       { \CT@arc@ \vline }
222       \futurelet \reserved@a
223       \@xhline
224     }

```

```

225     }
226 }

```

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.

```

227 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
228 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
229 {
230   \int_if_zero:nT \l_@@_first_col_int { \omit & }
231   \int_compare:nNnT { #1 } > 1
232     { \multispan { \int_eval:n { #1 - 1 } } & }
233   \multispan { \int_eval:n { #2 - #1 + 1 } }
234   {
235     \CT@arc@
236     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

237     \skip_horizontal:N \c_zero_dim
238 }

```

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`.

```

239 \everycr { }
240 \cr
241 \noalign { \skip_vertical:n { - \arrayrulewidth } }
242 }

```

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.

```

243 \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`.

```

244 { \@@_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*.

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_generate_variant:Nn \@@_cline_i:nn { e }
247 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
248 {
249   \tl_if_empty:nTF { #3 }
250     { \@@_cline_iii:w #1|#2-#2 \q_stop }
251     { \@@_cline_ii:w #1|#2-#3 \q_stop }
252 }
253 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
254 { \@@_cline_iii:w #1|#2-#3 \q_stop }
255 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
256 {

```

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).

```

257   \int_compare:nNnT { #1 } < { #2 }
258     { \multispan { \int_eval:n { #2 - #1 } } & }
259   \multispan { \int_eval:n { #3 - #2 + 1 } }
260   {
261     \CT@arc@

```

---

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

```

262     \leaders \hrule \@height \arrayrulewidth \hfill
263     \skip_horizontal:N \c_zero_dim
264 }

```

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

```

265 \peek_meaning_remove_ignore_spaces:NTF \cline
266 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
267 { \everycr { } \cr }
268 }

```

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

```

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

270 \cs_new_protected:Npn \@@_set_CTarc:n #1
271 {
272   \tl_if_blank:nF { #1 }
273   {
274     \tl_if_head_eq_meaning:nNTF { #1 } [
275       { \def \CT@arc@ { \color #1 } }
276       { \def \CT@arc@ { \color { #1 } } }
277     ]
278   }
279 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

280 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
281 {
282   \tl_if_head_eq_meaning:nNTF { #1 } [
283     { \def \CT@drsc@ { \color #1 } }
284     { \def \CT@drsc@ { \color { #1 } } }
285   ]

```

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

```

286 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
287 {
288   \tl_if_head_eq_meaning:nNTF { #2 } [
289     { #1 #2 }
290     { #1 { #2 } }
291   ]
292 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

293 \cs_new_protected:Npn \@@_color:n #1
294 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
295 \cs_generate_variant:Nn \@@_color:n { o }

```

```

296 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
297 {
298   \tl_set_rescan:Nno
299   #1
300   {
301     \char_set_catcode_other:N >
302     \char_set_catcode_other:N <
303   }
304   #1
305 }

```

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

```

306 \dim_new:N \l_@@_tmpc_dim
307 \dim_new:N \l_@@_tmpd_dim

308 \tl_new:N \l_@@_tmpc_tl
309 \tl_new:N \l_@@_tmpd_tl

310 \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.

```

311 \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).

```

312 \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.

```

313 \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.

```

314 \box_new:N \l_@@_the_array_box

```

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

```

315 \cs_new_protected:Npn \@@_qpoint:n #1
316 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

317 \bool_new:N \l_@@_tabular_bool

```

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

```

318 \bool_new:N \g_@@_delims_bool
319 \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.

```

320 \bool_new:N \l_@@_preamble_bool
321 \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.

```

322 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool

```

The following counter will count the environments {NiceMatrixBlock}.

```
323 \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.

```
324 \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).

```
325 \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.).

```
326 \dim_new:N \l_@@_col_width_dim
327 \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.

```
328 \int_new:N \g_@@_row_total_int
329 \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).

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

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

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

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

```
332 \tl_new:N \l_@@_hpos_cell_tl
333 \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`.

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

Idem for the mono-row blocks.

```
335 \dim_new:N \g_@@_blocks_ht_dim
336 \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}).

```
337 \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`.

```
338 \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.

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

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

```
340 \bool_new:N \l_@@_notes_detect_duplicates_bool
341 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
342 \bool_new:N \l_@@_initial_open_bool
343 \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.

```
344 \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).

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

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

```
346 \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).

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

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

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

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

```
349 \bool_new:N \g_@@_rotate_c_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.

```
350 \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`).

```
351 \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`.

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

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

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

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

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

```
355 \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 up it).

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`.

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

```
364 \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 `:`.

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

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

```
367 \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`).

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

```
369 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
370 {
```

```
371   \if_mode_math: \else:
```

```
372     \@@_fatal:n { Outside-math-mode }
```

```
373   \fi:
```

```
374 }
```

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.

```
375 \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”.

```
376 \colorlet { nicematrix-last-col } { . }
377 \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`).

```
378 \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*.

```
379 \str_new:N \g_@@_com_or_env_str
380 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

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

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

```
383 \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)`.

```
384 \cs_new:Npn \@@_full_name_env:
385 {
386   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
387   { command \space \c_backslash_str \g_@@_name_env_str }
388   { environment \space \{ \g_@@_name_env_str \} }
389 }
```

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

The argument is given by curryfication.

```
391 \cs_new_protected:Npn \@@_put_in_cell_after_hook:n
392 { \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.

```
393 \tl_new:N \g_@@_pre_code_before_tl
394 \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`.

```
395 \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.

```
396 \tl_new:N \g_@@_pre_code_after_tl
397 \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.

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

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

```
399 \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.

```
400 \int_new:N \l_@@_old_iRow_int
401 \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).

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

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

```
403 \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.

```
404 \bool_new:N \l_@@_X_columns_aux_bool
405 \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.

```
406 \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).

```
407 \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).

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

```
409 \tl_new:N \l_@@_code_before_tl
410 \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).

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

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

```
412 \dim_new:N \l_@@_x_initial_dim
413 \dim_new:N \l_@@_y_initial_dim
414 \dim_new:N \l_@@_x_final_dim
415 \dim_new:N \l_@@_y_final_dim

416 \dim_new:N \g_@@_dp_row_zero_dim
417 \dim_new:N \g_@@_ht_row_zero_dim
418 \dim_new:N \g_@@_ht_row_one_dim
419 \dim_new:N \g_@@_dp_ante_last_row_dim
420 \dim_new:N \g_@@_ht_last_row_dim
421 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
422 \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”.

```
423 \dim_new:N \g_@@_width_last_col_dim
424 \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.

```
425 \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!).

```
426 \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.

```
427 \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}`.

```
428 \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.

```
429 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
430 \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`).

```
431 \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.

```
432 \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).

```
433 \seq_new:N \g_@@_multicolumn_cells_seq
434 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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

```
435 \int_new:N \l_@@_multiplicity_int
436 \int_set:Nn \l_@@_multiplicity_int { 1 }
```

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

```
437 \int_new:N \g_@@_ddots_int
438 \int_new:N \g_@@_iddots_int
```

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

```
439 \dim_new:N \g_@@_delta_x_one_dim
440 \dim_new:N \g_@@_delta_y_one_dim
441 \dim_new:N \g_@@_delta_x_two_dim
442 \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).

```
443 \int_new:N \l_@@_row_min_int
444 \int_new:N \l_@@_row_max_int
445 \int_new:N \l_@@_col_min_int
446 \int_new:N \l_@@_col_max_int

447 \int_new:N \l_@@_initial_i_int
448 \int_new:N \l_@@_initial_j_int
449 \int_new:N \l_@@_final_i_int
450 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
451 \int_new:N \l_@@_segment_start_int
452 \int_new:N \l_@@_segment_end_int
```

---

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

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.

```
453 \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).

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

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

```
455 \tl_new:N \l_@@_draw_tl
```

```
456 \seq_new:N \l_@@_tikz_seq
```

```
457 \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`).

```
458 \str_new:N \l_@@_hpos_block_str
```

```
459 \str_set:Nn \l_@@_hpos_block_str { c }
```

```
460 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

```
461 \bool_new:N \l_@@_p_block_bool
```

```
462 \bool_new:N \l_@@_fix_vertex_bool
```

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

```
463 \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).

```
464 \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.

```
465 \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}`).

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

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

```
467 \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`).

```
468 \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.

```
469 \int_new:N \l_@@_first_row_int
470 \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.

```
471 \int_new:N \l_@@_first_col_int
472 \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).

```
473 \int_new:N \l_@@_last_row_int
474 \int_set:Nn \l_@@_last_row_int { -2 }
```

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
476 \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.

```
477 \int_new:N \l_@@_last_col_int
478 \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}
```

---

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

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.

```
479 \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`).

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

## Some utilities

```
481 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
482 {
```

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

```
483 \def \l_tmpa_tl { #1 }
484 \def \l_tmpb_tl { #2 }
485 }
```

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`.

```
486 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
487 {
488   \clist_if_in:NnF #1 { all }
489   {
490     \clist_clear:N \l_tmpa_clist
491     \clist_map_inline:Nn #1
492     {
493       \tl_if_head_eq_meaning:nNTF { ##1 } -
494       {
```

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.

```
495       \int_if_zero:nF { #2 }
496       {
497         \clist_put_right:Ne \l_tmpa_clist
498         { \int_eval:n { #2 + (##1) + 1 } }
499       }
500     }
501   }
```

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

```
502       \tl_if_in:nnTF { ##1 } { - }
503       { \@@_cut_on_hyphen:w ##1 \q_stop }
504       {
```

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

```
505       \def \l_tmpa_tl { ##1 }
506       \def \l_tmpb_tl { ##1 }
507     }
508     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
509     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
510   }
511 }
512 \tl_set_eq:NN #1 \l_tmpa_clist
513 }
514 }
```

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.

```

515 \AtBeginDocument
516 {
517     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
518     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
519 }

```

## 5 The command `\tabularnote`

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

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

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.

```

520 \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`.

```

521 \int_new:N \g_@@_tabularnote_int
522 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

```

---

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

```

523 \seq_new:N \g_@@_notes_seq
524 \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.

```

525 \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.

```

526 \seq_new:N \l_@@_notes_labels_seq
527 \newcounter { nicematrix_draft }
528 \cs_new_protected:Npn \@@_notes_format:n #1
529 {
530   \setcounter { nicematrix_draft } { #1 }
531   \@@_notes_style:n { nicematrix_draft }
532 }

```

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

```

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

```

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

```

534 \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`.

```

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

```

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

```

536 \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`).

```

537 \AtBeginDocument
538 {
539   \IfPackageLoadedTF { enumitem }
540   {

```

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.

```

541     \newlist { tabularnotes } { enumerate } { 1 }
542     \setlist [ tabularnotes ]
543     {
544       topsep = \c_zero_dim ,
545       noitemsep ,
546       leftmargin = * ,
547       align = left ,
548       labelsep = \c_zero_dim ,
549       label =
550         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
551     }
552     \newlist { tabularnotes* } { enumerate* } { 1 }
553     \setlist [ tabularnotes* ]
554     {
555       afterlabel = \nobreak ,

```

```

556         itemjoin = \quad ,
557         label =
558             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
559     }

```

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

```

560     \NewDocumentCommand { \tabularnote } { o m }
561     {
562         \bool_lazy_or:nnT \l_@@_in_env_bool { \cs_if_exist_p:N \@capttype }
563         {
564             \bool_lazy_and:nnTF \l_@@_in_env_bool { ! \l_@@_tabular_bool }
565             { \@@_error:n { tabularnote~forbidden } }
566             {

```

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

```

567                 \bool_if:NTF \l_@@_in_caption_bool
568                 {
569                     \tl_if_novalue:nTF { #1 }
570                     { \@@_tabularnote_caption:nn { #1 } }
571                     { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
572                 }
573                 {
574                     \tl_if_novalue:nTF { #1 }
575                     { \@@_tabularnote:nn { #1 } }
576                     { \@@_tabularnote:nn { \exp_not:n { #1 } } }
577                 }
578                 { #2 }
579             }
580         }
581     }
582 }
583 {
584     \NewDocumentCommand \tabularnote { o m }
585     { \@@_err_enumitem_not_loaded: }
586 }
587 }
588 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
589 {
590     \@@_error_or_warning:n { enumitem~not~loaded }
591     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
592 }
593 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
594 { \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`.

```

595 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
596 {

```

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).

```

597     \int_zero:N \l_tmpa_int
598     \bool_if:NT \l_@@_notes_detect_duplicates_bool
599     {

```

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

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

If the user have used `\tabulernote` 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@tabulernote`.

```

600      \int_zero:N \l_tmpb_int
601      \seq_map_indexed_inline:Nn \g_@@_notes_seq
602      {
603          \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
604          \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
605              {
606                  \tl_if_novalue:nTF { #1 }
607                      { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
608                      { \int_set:Nn \l_tmpa_int { ##1 } }
609                  \seq_map_break:
610              }
611          }
612      \int_if_zero:nF \l_tmpa_int
613      { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
614  }
615  \int_if_zero:nT \l_tmpa_int
616  {
617      \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
618      \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
619  }
620  \seq_put_right:Ne \l_@@_notes_labels_seq
621  {
622      \tl_if_novalue:nTF { #1 }
623      {
624          \@@_notes_format:n
625          {
626              \int_eval:n
627              {
628                  \int_if_zero:nTF \l_tmpa_int
629                      \c@tabulernote
630                      \l_tmpa_int
631              }
632          }
633      }
634      { #1 }
635  }
636  \peek_meaning:NF \tabulernote
637  {

```

If the following token is *not* a `\tabulernote`, we have finished the sequence of successive commands `\tabulernote` 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`.

```

638      \hbox_set:Nn \l_tmpa_box
639      {

```

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

```

640          \@@_notes_label_in_tabular:n
641          { \seq_use:Nn \l_@@_notes_labels_seq { , } }
642      }

```

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

```

643      \int_gdecr:N \c@tabulernote
644      \int_set_eq:NN \l_tmpa_int \c@tabulernote

```

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

```

645 \int_gincr:N \g_@@_tabularnote_int
646 \refstepcounter { tabularnote }
647 \int_compare:nNnT \l_tmpa_int = \c@tabularnote
648 { \int_gincr:N \c@tabularnote }
649 \seq_clear:N \l_@@_notes_labels_seq
650 \bool_lazy_or:nnTF
651 { \str_if_eq_p:ee c \l_@@_hpos_cell_tl }
652 { \str_if_eq_p:ee r \l_@@_hpos_cell_tl }
653 {
654 \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).

```

655 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
656 }
657 { \box_use:N \l_tmpa_box }
658 }
659 }

```

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`.

```

660 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
661 {
662 \bool_if:NTF \g_@@_caption_finished_bool
663 {
664 \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
665 { \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`!

```

666 \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
667 { \@@_error:n { Identical-notes-in-caption } }
668 }
669 {

```

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

```

670 \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
671 {

```

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.

```

672 \bool_gset_true:N \g_@@_caption_finished_bool
673 \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
674 \int_gzero:N \c@tabularnote
675 }
676 { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
677 }

```

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!

```

678 \tl_if_noval:nt { #1 } { \int_gincr:N \c@tabularnote }
679 \seq_put_right:Ne \l_@@_notes_labels_seq
680 {
681 \tl_if_noval:ntTF { #1 }

```

```

682         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
683         { #1 }
684     }
685     \peek_meaning:NF \tabularnote
686     {
687         \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
688         \seq_clear:N \l_@@_notes_labels_seq
689     }
690 }
691 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
692 { \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.

```

693 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
694 {
695     \begin { pgfscope }
696     \pgfset
697     {
698         inner~sep = \c_zero_dim ,
699         minimum~size = \c_zero_dim
700     }
701     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
702     \pgfnode
703     { rectangle }
704     { center }
705     {
706         \vbox_to_ht:nn
707         { \dim_abs:n { #5 - #3 } }
708         {
709             \vfill
710             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { } { }
711         }
712     }
713     { #1 }
714     { }
715     \end { pgfscope }
716 }

```

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.

```

717 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
718 {
719     \begin { pgfscope }
720     \pgfset
721     {
722         inner~sep = \c_zero_dim ,
723         minimum~size = \c_zero_dim
724     }
725     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
726     \pgfpointdiff { #3 } { #2 }
727     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
728     \pgfnode
729     { rectangle }
730     { center }

```

```

731     {
732         \vbox_to_ht:nn
733         { \dim_abs:n \l_tmpb_dim }
734         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
735     }
736     { #1 }
737     { }
738 \end { pgfscope }
739 }

```

## 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).

```

740 \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.

```

741 \dim_new:N \l_@@_xdots_inter_dim
742 \AtBeginDocument
743 { \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).

```

744 \dim_new:N \l_@@_xdots_shorten_start_dim
745 \dim_new:N \l_@@_xdots_shorten_end_dim
746 \AtBeginDocument
747 {
748     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
749     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
750 }

```

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.

```

751 \dim_new:N \l_@@_xdots_radius_dim
752 \AtBeginDocument
753 { \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.

```

754 \tl_new:N \l_@@_xdots_line_style_tl
755 \tl_const:Nn \c_@@_standard_tl { standard }
756 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

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

```
757 \bool_new:N \l_@@_light_syntax_bool
758 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

```
759 \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).

```
760 \bool_new:N \l_@@_auto_columns_width_bool
```

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

```
761 \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.

```
762 \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”.

```
763 \bool_new:N \l_@@_medium_nodes_bool
764 \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).

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

```
766 \keys_define:nn { nicematrix / xdots }
767 {
```

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.).

```
768   nullify .bool_set:N = \l_@@_nullify_dots_bool ,
769   brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
770   brace-shift+ .code:n =
771     \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
772   brace-shift+ .value_required:n = true ,
773   brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
774   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
775   shorten-start .code:n =
776     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
777   shorten-start .value_required:n = true ,
778   shorten-start+ .code:n =
779     \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
780   shorten-start~+ .meta:n = { shorten-start += #1 } ,
781   shorten-start+ .value_required:n = true ,
782   shorten-end .code:n =
783     \AtBeginDocument { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
784   shorten-end .value_required:n = true ,
785   shorten-end+ .code:n =
786     \AtBeginDocument { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
787   shorten-end+ .value_required:n = true ,
788   shorten-end~+ .meta:n = { shorten-end += #1 } ,
789   shorten .code:n =
```

```

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

```

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

```

822 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
823 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
824 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).

```

825 draw-first .code:n = \prg_do_nothing: ,
826 unknown .code:n =
827     \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
828 }

```

```

829 \keys_define:nn { nicematrix / trees }
830 {
831     color.tl_set:N = \l_@@_trees_color_tl ,
832     color .value_required:n = true ,
833     line-width .dim_set:N = \l_@@_trees_line_width_dim ,
834     rounded-corners .dim_set:N = \l_@@_trees_rounded_corners_dim ,
835     rounded-corners .default:n = 2 pt ,
836     unknown .code:n =
837         \@@_unknown_key:nn { nicematrix / trees } { Unknown~key~for~trees }
838 }

```

```

839 \keys_define:nn { nicematrix / rules }
840 {
841     fix-vertex .code:n =

```

```

842     \bool_set_true:N \l_@@_fix_vertex_bool
843     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-h } { active }
844     \socket_assign_plug:nn { nicematrix / draw-at-odd-vertex-v } { active } ,
845     color .tl_set:N = \l_@@_rules_color_tl ,
846     color .value_required:n = true ,
847     width .dim_set:N = \arrayrulewidth ,
848     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
849 }

```

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

```

850 \keys_define:nn { nicematrix / Global }
851 {
852     fix-vertex .value_forbidden:n = true ,
853     brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
854     brace-shift+ .code:n =
855         \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
856     brace-shift+ .value_required:n = true ,
857     brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
858     caption-above .code:n = \@@_error_or_warning:n { caption-above-in-env } ,
859     show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
860     color-inside .code:n = \@@_fatal:n { key~color~inside } ,
861     colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
862     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
863     &-in-blocks .meta:n = ampersand-in-blocks ,
864     no-cell-nodes .code:n =
865         \bool_set_true:N \l_@@_no_cell_nodes_bool
866         \cs_set_protected:Npn \@@_node_cell:
867             { \set@color \box_use_drop:N \l_@@_cell_box } ,
868     no-cell-nodes .value_forbidden:n = true ,

```

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

```

869     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
870     rounded-corners .default:n = 4 pt ,
871     custom-line .code:n = \@@_custom_line:n { #1 } ,
872     default-line .code:n = \@@_default_line:n { #1 } ,
873     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
874     rules .value_required:n = true ,
875     trees .code:n = \keys_set:nn { nicematrix / trees } { #1 } ,
876     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`.

```

877     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
878     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
879     cell-space-top-limit+ .code:n =
880         \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
881     cell-space-top-limit+ .value_required:n = true ,
882     cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
883     cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
884     cell-space-bottom-limit+ .code:n =
885         \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
886     cell-space-bottom-limit+ .value_required:n = true ,
887     cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
888     cell-space-limits .meta:n =
889         {
890             cell-space-top-limit = #1 ,
891             cell-space-bottom-limit = #1 ,
892         } ,
893     cell-space-limits .value_required:n = true ,
894     cell-space-limits+ .meta:n =

```

```

895     {
896         cell-space-top-limit += #1 ,
897         cell-space-bottom-limit += #1 ,
898     } ,
899     cell-space-limits+ .value_required:n = true ,
900     cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
901     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
902     light-syntax .code:n =
903         \bool_set_true:N \l_@@_light_syntax_bool
904         \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
905     light-syntax .value_forbidden:n = true ,
906     light-syntax-expanded .code:n =
907         \bool_set_true:N \l_@@_light_syntax_bool
908         \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
909     light-syntax-expanded .value_forbidden:n = true ,

```

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

```

910     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
911     end-of-row .value_required:n = true ,
912     end-of-row .initial:n = ; ,
913
914     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
915     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
916     last-row .int_set:N = \l_@@_last_row_int ,
917     last-row .default:n = -1 ,
918
919     code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
920     code-for-first-col .value_required:n = true ,
921     code-for-first-col+ .code:n =
922         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
923     code-for-first-col+ .value_required:n = true ,
924     code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,
925
926     code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
927     code-for-last-col .value_required:n = true ,
928     code-for-last-col+ .code:n =
929         { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
930     code-for-last-col+ .value_required:n = true ,
931     code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,
932
933     code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
934     code-for-first-row .value_required:n = true ,
935     code-for-first-row+ .code:n =
936         { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
937     code-for-first-row+ .value_required:n = true ,
938     code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,
939
940     code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
941     code-for-last-row .value_required:n = true ,
942     code-for-last-row+ .code:n =
943         { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
944     code-for-last-row+ .value_required:n = true ,
945     code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,
946
947     hlines .clist_set:N = \l_@@_hlines_clist ,
948     hlines .default:n = all ,
949     vlines .clist_set:N = \l_@@_vlines_clist ,
950     vlines .default:n = all ,
951
952     vlines-in-sub-matrix .code:n =
953     {
954         \tl_if_single_token:nTF { #1 }
955         {

```

```

956      \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
957      { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

958      { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
959    }
960    { \@@_error:n { One~letter~allowed } }
961  } ,
962  vlines-in-sub-matrix .value_required:n = true ,
963  hvlines .code:n =
964  {
965    \bool_set_true:N \l_@@_hvlines_bool
966    \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
967    \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
968  } ,
969  hvlines .value_forbidden:n = true ,
970  hvlines-except-borders .code:n =
971  {
972    \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
973    \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
974    \bool_set_true:N \l_@@_hvlines_bool
975    \bool_set_true:N \l_@@_except_borders_bool
976  } ,
977  hlines-except-borders .value_forbidden:n = true ,
978  hlines-except-borders .code:n =
979  {
980    \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
981    \bool_set_true:N \l_@@_hlines_bool
982    \bool_set_true:N \l_@@_except_borders_bool
983  } ,
984  hlines-except-borders .value_forbidden:n = true ,
985  parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,
986  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.

```

987  renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
988  renew-dots .value_forbidden:n = true ,
989  nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
990  create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
991  create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
992  create-extra-nodes .meta:n =
993  { create-medium-nodes , create-large-nodes } ,
994  left-margin .dim_set:N = \l_@@_left_margin_dim ,
995  left-margin .default:n = \arraycolsep ,
996  right-margin .dim_set:N = \l_@@_right_margin_dim ,
997  right-margin .default:n = \arraycolsep ,
998  margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
999  margin .default:n = \arraycolsep ,
1000  extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
1001  extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
1002  extra-margin .meta:n =
1003  { extra-left-margin = #1 , extra-right-margin = #1 } ,
1004  extra-margin .value_required:n = true ,
1005  respect-arraystretch .code:n =
1006  \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
1007  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.

```

1008  pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
1009  pgf-node-code .value_required:n = true ,
1010 }

```

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

```

1011 \keys_define:nn { nicematrix / environments }
1012 {
1013   draw-trees-in-col .code:n =
1014     \clist_gput_right:Nn \g_@@_col_with_trees_clist { #1 } ,
1015   draw-trees-in-col .value_required:n = true ,
1016   create-blocks-in-col .code:n =
1017     \clist_gput_right:Nn \g_@@_cbic_clist { #1 } ,
1018   create-blocks-in-col .value_required:n = true ,
1019   corners .clist_set:N = \l_@@_corners_clist ,
1020   corners .default:n = { NW , SW , NE , SE } ,
1021   code-before .code:n =
1022     {
1023       \tl_if_empty:nF { #1 }
1024       {
1025         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1026         \bool_set_true:N \l_@@_code_before_bool
1027       }
1028     } ,
1029   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}`.

```

1030   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
1031   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
1032   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).

```

1033   baseline .tl_set:N = \l_@@_baseline_tl ,
1034   baseline .value_required:n = true ,
1035   baseline .initial:n = c ,
1036   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1037   \str_if_eq:eeTF { #1 } { auto }
1038   { \bool_set_true:N \l_@@_auto_columns_width_bool }
1039   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1040   columns-width .value_required:n = true ,
1041   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.

```

1042   \legacy_if:nF { measuring@ }
1043   {
1044     \str_set:Ne \l_@@_name_str { #1 }
1045     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1046     { \@@_err_duplicate_names:n { #1 } }
1047     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1048   } ,
1049   name .value_required:n = true ,
1050   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1051   code-after .value_required:n = true ,
1052 }

1053 \cs_set:Npn \@@_err_duplicate_names:n #1
1054 { \@@_error:nn { Duplicate-name } { #1 } }

1055 \keys_define:nn { nicematrix / notes }
1056 {
1057   no-print .bool_set:N = \l_@@_notes_no_print_bool ,

```

```

1058 para .bool_set:N = \l_@@_notes_para_bool ,
1059 code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1060 code-before .value_required:n = true ,
1061 code-before+ .code:n =
1062   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1063 code-before+ .value_required:n = true ,
1064 code-before~+ .code:n =
1065   \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1066 code-before~+ .value_required:n = true ,
1067 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1068 code-after .value_required:n = true ,
1069 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1070 style .cs_set:Np = \@@_notes_style:n #1 ,
1071 style .value_required:n = true ,
1072 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1073 label-in-tabular .value_required:n = true ,
1074 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1075 label-in-list .value_required:n = true ,
1076 enumitem-keys .code:n =
1077   {
1078     \AtBeginDocument
1079     {
1080       \IfPackageLoadedT { enumitem }
1081       { \setlist* [ tabularnotes ] { #1 } }
1082     }
1083   } ,
1084 enumitem-keys .value_required:n = true ,
1085 enumitem-keys-para .code:n =
1086   {
1087     \AtBeginDocument
1088     {
1089       \IfPackageLoadedT { enumitem }
1090       { \setlist* [ tabularnotes* ] { #1 } }
1091     }
1092   } ,
1093 enumitem-keys-para .value_required:n = true ,
1094 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1095 unknown .code:n =
1096   \@@_unknown_key:nn { nicematrix / notes } { Unknown-key-for-notes }
1097 }

```

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.

```

1098 \keys_define:nn { nicematrix / delimiters }
1099 {
1100   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1101   color .tl_set:N = \l_@@_delimiters_color_tl ,
1102   color .value_required:n = true ,
1103 }

```

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

```

1104 \keys_define:nn { nicematrix }
1105 {
1106   NiceMatrixOptions .inherit:n =
1107     { nicematrix / Global } ,
1108   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

1109 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1110 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1111 NiceMatrixOptions / trees .inherit:n = nicematrix / trees ,
1112 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1113 SubMatrix / rules .inherit:n = nicematrix / rules ,
1114 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1115 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1116 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1117 NiceMatrix .inherit:n =
1118 {
1119     nicematrix / Global ,
1120     nicematrix / environments ,
1121 } ,
1122 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1123 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1124 NiceMatrix / trees .inherit:n = nicematrix / trees ,
1125 NiceTabular .inherit:n =
1126 {
1127     nicematrix / Global ,
1128     nicematrix / environments
1129 } ,
1130 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1131 NiceTabular / rules .inherit:n = nicematrix / rules ,
1132 NiceTabular / notes .inherit:n = nicematrix / notes ,
1133 NiceTabular / trees .inherit:n = nicematrix / trees ,
1134 NiceArray .inherit:n =
1135 {
1136     nicematrix / Global ,
1137     nicematrix / environments ,
1138 } ,
1139 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1140 NiceArray / rules .inherit:n = nicematrix / rules ,
1141 NiceArray / trees .inherit:n = nicematrix / trees ,
1142 pNiceArray .inherit:n =
1143 {
1144     nicematrix / Global ,
1145     nicematrix / environments ,
1146 } ,
1147 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1148 pNiceArray / rules .inherit:n = nicematrix / rules ,
1149 pNiceArray / trees .inherit:n = nicematrix / trees
1150 }

```

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

```

1151 \keys_define:nn { nicematrix / NiceMatrixOptions }
1152 {
1153     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1154     delimiters / color .value_required:n = true ,
1155     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1156     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1157     delimiters .value_required:n = true ,
1158     width .dim_set:N = \l_@@_width_dim ,
1159     last-col .code:n =
1160         \tl_if_empty:nF { #1 }
1161         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1162         \int_zero:N \l_@@_last_col_int ,
1163     small .bool_set:N = \l_@@_small_bool ,
1164     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.

```

1165     renew-matrix .code:n = \@@_renew_matrix: ,
1166     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}`.

```
1167     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.

```
1168     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)`.

```
1169     \str_if_eq:eeTF { #1 } { auto }
1170     { \@@_error:n { Option~auto~for~columns~width } }
1171     { \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.

```
1172     allow-duplicate-names .code:n =
1173     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1174     allow-duplicate-names .value_forbidden:n = true ,
1175     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1176     notes .value_required:n = true ,
1177     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1178     sub-matrix .value_required:n = true ,
1179     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1180     matrix / columns-type .value_required:n = true ,
1181     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1182     unknown .code:n =
1183     \@@_unknown_key:nn
1184     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1185     { Unknown-key-for~NiceMatrixOptions }
1186 }
```

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

```
1187 \NewDocumentCommand \NiceMatrixOptions { m }
1188 { \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.

```
1189 \keys_define:nn { nicematrix / NiceMatrix }
1190 {
1191     last-col .code:n = \tl_if_empty:nTF { #1 }
1192     {
1193         \bool_set_true:N \l_@@_last_col_without_value_bool
1194         \int_set:Nn \l_@@_last_col_int { -1 }
1195     }
1196     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1197     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1198     columns-type .value_required:n = true ,
1199     l .meta:n = { columns-type = l } ,
1200     r .meta:n = { columns-type = r } ,
1201     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1202     delimiters / color .value_required:n = true ,
1203     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1204     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1205     delimiters .value_required:n = true ,
1206     small .bool_set:N = \l_@@_small_bool ,
1207     small .value_forbidden:n = true ,
1208     unknown .code:n =
```

```

1209 \@@_unknown_key:nn
1210 { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
1211 { Unknown-key-for-NiceMatrix }
1212 }

```

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

```

1213 \keys_define:nn { nicematrix / NiceArray }
1214 {

```

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.

```

1215 small .bool_set:N = \l_@@_small_bool ,
1216 small .value_forbidden:n = true ,
1217 last-col .code:n = \tl_if_empty:nF { #1 }
1218 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1219 \int_zero:N \l_@@_last_col_int ,
1220 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1221 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1222 unknown .code:n =
1223 \@@_unknown_key:nn
1224 { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1225 { Unknown-key-for-NiceArray }
1226 }

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

```

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

```

1248 \keys_define:nn { nicematrix / NiceTabular }
1249 {

```

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.

```

1250 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1251 \bool_set_true:N \l_@@_width_used_bool ,
1252 width .value_required:n = true ,
1253 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1254 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1255 tabularnote .value_required:n = true ,
1256 caption .tl_set:N = \l_@@_caption_tl ,
1257 caption .value_required:n = true ,

```

```

1258 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1259 short-caption .value_required:n = true ,
1260 label .tl_set:N = \l_@@_label_tl ,
1261 label .value_required:n = true ,
1262 last-col .code:n = \tl_if_empty:nF { #1 }
1263 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1264 \int_zero:N \l_@@_last_col_int ,
1265 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1266 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1267 unknown .code:n =
1268 \@@_unknown_key:nn
1269 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1270 { Unknown-key-for-NiceTabular }
1271 }

```

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
```

```

1272 \keys_define:nn { nicematrix / CodeAfter }
1273 {
1274   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1275   delimiters / color .value_required:n = true ,
1276   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1277   rules .value_required:n = true ,
1278   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1279   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1280   sub-matrix .value_required:n = true ,
1281   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1282 }

```

## 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}`).

```

1283 \cs_new_protected:Npn \@@_cell_begin:
1284 {

```

`\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.

```
1285 \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

```
1286 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1287 \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
1288 \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:nNt { \c@jCol } = { 1 }
{ \int_compare:nNt \l_@@_first_col_int = { 1 } { \@@_begin_of_row: } }
```

We will use a version a little more efficient.

```
1289 \if_int_compare:w \c@jCol = \c_one_int
1290 \if_int_compare:w \l_@@_first_col_int = \c_one_int
1291 \@@_begin_of_row:
1292 \fi:
1293 \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:`.

```
1294 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1295 \@@_tuning_not_tabular_begin:
1296 \@@_tuning_exterior_rows:
1297 \g_@@_row_style_tl
1298 }
1299 \cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of L3.

```
\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.

```
1300 \cs_new_protected:Npn \@@_tuning_first_last_row:
1301 {
1302 \if_int_compare:w \c@iRow = \c_zero_int
1303 \if_int_compare:w \c@jCol > \c_zero_int
1304 \l_@@_code_for_first_row_tl
1305 \xglobal \colorlet { nicematrix-first-row } { . }
1306 \fi:
1307 \else:
1308 \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1309 \fi:
1310 }
```

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

```
\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNt { \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.

```
1311 \cs_new_protected:Npn \@@_tuning_last_row:
1312 {
1313 \if_int_compare:w \c@iRow = \l_@@_last_row_int
```

```

1314     \l_@@_code_for_last_row_tl
1315     \xglobal \colorlet { nicematrix-last-row } { . }
1316     \fi:
1317 }

```

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

```

1318 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1319 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1320 {
1321     \m@th
1322     $ % $

```

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

```

1323     \@@_tuning_key_small:
1324 }
1325 \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.

```

1326 \cs_new_protected:Npn \@@_begin_of_row:
1327 {
1328     \int_gincr:N \c@iRow
1329     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1330     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1331     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1332     \pgfpicture
1333     \pgfrememberpicturepositiononpagetrue
1334     \pgfcoordinate
1335     { \@@_env: - row - \int_use:N \c@iRow - base }
1336     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1337     \str_if_empty:NF \l_@@_name_str
1338     {
1339         \pgfnodelalias
1340         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1341         { \@@_env: - row - \int_use:N \c@iRow - base }
1342     }
1343     \endpgfpicture
1344 }

```

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 L3.

```

\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.

```

1345 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1346 {
1347   \if_int_compare:w \c@iRow = \c_zero_int
1348     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1349     \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1350     \fi:
1351     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1352     \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1353     \fi:
1354   \else:
1355     \if_int_compare:w \c@iRow = \c_one_int
1356     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1357     \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1358     \fi:
1359   \fi:
1360 \fi:
1361 }

1362 \cs_new_protected:Npn \@@_rotate_cell_box:
1363 {
1364   \box_rotate:Nn \l_@@_cell_box { 90 }
1365   \bool_if:NTF \g_@@_rotate_c_bool
1366   {
1367     \hbox_set:Nn \l_@@_cell_box
1368     {
1369       \m@th
1370       $ % $
1371       \vcenter { \box_use:N \l_@@_cell_box }
1372       $ % $
1373     }
1374   }
1375   {
1376     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1377     {
1378       \vbox_set_top:Nn \l_@@_cell_box
1379       {
1380         \vbox_to_zero:n { }
1381         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1382         \box_use:N \l_@@_cell_box
1383       }
1384     }
1385   }
1386   \bool_gset_false:N \g_@@_rotate_bool
1387   \bool_gset_false:N \g_@@_rotate_c_bool
1388 }

```

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.

```

1389 \cs_set_protected:Npn \@@_adjust_size_box:
1390 {
1391     \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1392         \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1393             \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1394         \fi:
1395         \global \g_@@_blocks_wd_dim = \c_zero_dim
1396     \fi:
1397     \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1398         \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1399             \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1400         \fi
1401         \global \g_@@_blocks_dp_dim = \c_zero_dim
1402     \fi:
1403     \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1404         \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1405             \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1406         \fi:
1407         \global \g_@@_blocks_ht_dim = \c_zero_dim
1408     \fi:
1409 }
1410 \cs_new_protected:Npn \@@_cell_end:
1411 {

```

The following command is nullified in the tabulars.

```

1412 \@@_tuning_not_tabular_end:
1413 \hbox_set_end:
1414 \@@_cell_end_i:
1415 }

```

```

\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:nNt { \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.

```

1416 \cs_new_protected:Npn \@@_cell_end_i:
1417 {

```

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.

```

1418     \g_@@_cell_after_hook_tl
1419     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1420     \@@_adjust_size_box:
1421     \box_set_ht:Nn \l_@@_cell_box
1422     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1423     \box_set_dp:Nn \l_@@_cell_box
1424     { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

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

```

1425     \@@_update_max_cell_width:

```

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

```

1426     \@@_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.

```

1427     \bool_if:NTF \g_@@_empty_cell_bool
1428     { \box_use_drop:N \l_@@_cell_box }
1429     {
1430         \bool_if:NTF \g_@@_not_empty_cell_bool
1431         \@@_print_node_cell:
1432         {
1433             \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1434             \@@_print_node_cell:
1435             \else:
1436                 \box_use_drop:N \l_@@_cell_box
1437             \fi:

```

```

1438     }
1439   }
1440   \if_int_compare:w \c@jCol > \g_@@_col_total_int
1441     \global \g_@@_col_total_int = \c@jCol
1442   \fi:
1443   \global \let \g_@@_empty_cell_bool \c_false_bool
1444   \global \let \g_@@_not_empty_cell_bool \c_false_bool
1445 }

```

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:

```

1446 \cs_new_protected:Npn \@@_update_max_cell_width:
1447 {
1448   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1449     \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1450   \fi:
1451 }

```

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`).

```

1452 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1453 {
1454   \@@_math_toggle:
1455   \hbox_set_end:
1456   \bool_if:NF \g_@@_rotate_bool
1457     {
1458       \hbox_set:Nn \l_@@_cell_box
1459       {
1460         \makebox [ \l_@@_col_width_dim ] [ s ]
1461         { \hbox_unpack_drop:N \l_@@_cell_box }
1462       }
1463     }
1464   \@@_cell_end_i:
1465 }

```

```

1466 \pgfset
1467 {
1468   nicematrix / cell-node /.style =
1469   {
1470     inner~sep = \c_zero_dim ,
1471     minimum~width = \c_zero_dim
1472   }
1473 }

```

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.

```

1474 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1475 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1476 {
1477   \use:c
1478   {

```

```

1479     __siunitx_table_align_
1480     \bool_if:NTF \l__siunitx_table_text_bool
1481     \l__siunitx_table_align_text_tl
1482     \l__siunitx_table_align_number_str
1483     :n
1484   }
1485   { #1 }
1486 }

```

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!

```

1487 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1488 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1489 {
1490   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1491   \hbox:n
1492   {
1493     \pgfsys@markposition
1494     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1495   }
1496   #1
1497   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1498   \hbox:n
1499   {
1500     \pgfsys@markposition
1501     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1502   }
1503 }

1504 \cs_new_protected:Npn \@@_print_node_cell:
1505 {
1506   \socket_use:nn { nicematrix / siunitx-wrap }
1507   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1508 }

```

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

```

1509 \cs_new_protected:Npn \@@_node_cell:
1510 {
1511   \pgfpicture
1512   \pgfsetbaseline \c_zero_dim
1513   \pgfrememberpicturepositiononpagetrue
1514   \pgfset { nicematrix / cell-node }
1515   \pgfnode
1516   { rectangle }
1517   { base }
1518   {

```

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).

```

1519   \sys_if_engine_xetex:T { \set@color }
1520   \box_use:N \l_@@_cell_box
1521 }
1522 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1523 { \l_@@_pgf_node_code_tl }
1524 \str_if_empty:NF \l_@@_name_str
1525 {
1526   \pgfnodealias
1527   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1528   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }

```

```

1529     }
1530     \endpgfpicture
1531 }

```

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).

```

1532 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1533 {
1534     \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1535     { g_@@_ #2 _ lines _ tl }
1536     {
1537         \use:c { @@ _ draw _ #2 : nnn }
1538         { \int_use:N \c@iRow }
1539         { \int_use:N \c@jCol }
1540         { \exp_not:n { #3 } }
1541     }
1542 }

1543 \cs_new_protected:Npn \@@_array:n
1544 {
1545     \dim_set:Nn \col@sep
1546     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1547     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1548     { \def \@halignto { } }
1549     { \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`.

```

1550     \@tabarray

```

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

```

1551     [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1552 }
1553 \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`.

```

1554 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign

```

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

```

1555 \cs_new_protected:Npn \@@_create_row_node:
1556 {
1557   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1558   {
1559     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1560     \@@_create_row_node_i:
1561   }
1562 }
1563 \cs_new_protected:Npn \@@_create_row_node_i:
1564 {

```

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

```

1565   \hbox
1566   {
1567     \bool_if:NT \l_@@_code_before_bool
1568     {
1569       \vtop
1570       {
1571         \skip_vertical:N 0.5\arrayrulewidth
1572         \pgfsys@markposition
1573         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1574         \skip_vertical:N -0.5\arrayrulewidth
1575       }
1576     }
1577     \pgfpicture
1578     \pgfrememberpicturepositiononpagetrue
1579     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1580     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1581     \str_if_empty:NF \l_@@_name_str
1582     {
1583       \pgfnodealias
1584       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1585       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1586     }
1587     \endpgfpicture
1588   }
1589 }

```

```

1590 \cs_new_protected:Npn \@@_in_everycr:
1591 {
1592   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1593   \tbl_update_cell_data_for_next_row:
1594   \int_gzero:N \c@jCol
1595   \bool_gset_false:N \g_@@_after_col_zero_bool
1596   \bool_if:NF \g_@@_row_of_col_done_bool
1597   {
1598     \@@_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).

```

1599   \clist_if_empty:NF \l_@@_hlines_clist
1600   {
1601     \str_if_eq:eeF \l_@@_hlines_clist { all }
1602     {
1603       \clist_if_in:NiT
1604       \l_@@_hlines_clist
1605       { \int_eval:n { \c@iRow + 1 } }
1606     }
1607   }

```

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.

```

1608         \int_compare:nNnT \c@iRow > { -1 }
1609         {
1610             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1611             { \hrule height \arrayrulewidth width \c_zero_dim }
1612         }
1613     }
1614 }
1615 }
1616 }

```

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

```

1617 \cs_set_protected:Npn \@@_renew_dots:
1618 {
1619     \cs_set_eq:NN \ldots \@@_Ldots:
1620     \cs_set_eq:NN \cdots \@@_Cdots:
1621     \cs_set_eq:NN \vdots \@@_Vdots:
1622     \cs_set_eq:NN \ddots \@@_Ddots:
1623     \cs_set_eq:NN \iddots \@@_Iddots:
1624     \cs_set_eq:NN \dots \@@_Ldots:
1625     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1626 }

```

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 <sup>5</sup>.

```

1627 \AtBeginDocument
1628 {
1629     \IfPackageLoadedTF { booktabs }
1630     {
1631         \cs_new_protected:Npn \@@_patch_booktabs:
1632         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1633     }
1634     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1635 }

```

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

```

1636 \cs_new_protected:Npn \@@_some_initialization:
1637 {
1638     \@@_everycr:
1639     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1640     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1641     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1642     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1643     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1644     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1645 }

```

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

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

`\@@_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`.

```
1646 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1647 {
```

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

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

```
1648 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1649 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1650 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq
```

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

```
1651 \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.

```
1652 \fp_gzero:N \g_@@_total_X_weight_fp
1653 \bool_gset_false:N \g_@@_V_of_X_bool

1654 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1655 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1656 \@@_patch_booktabs:
1657 \box_clear_new:N \l_@@_cell_box
1658 \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}`).

```
1659 \bool_if:NT \l_@@_small_bool
1660 {

1661 \def \arraystretch { 0.47 }
1662 \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1663 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1664 }
```

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!

```
1665 \bool_if:NT \g_@@_create_cell_nodes_bool
1666 {
1667 \tl_put_right:Nn \@@_begin_of_row:
1668 {
1669 \pgfsys@markposition
1670 { \@@_env: - row - \int_use:N \c@iRow - base }
1671 }
1672 \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1673 }
```

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`.

```

1674 \def \ar@ialign
1675 {
1676   \tbl_init_cell_data_for_table:
1677   \@@_some_initialization:
1678   \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`.

```

1679   \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1680   \halign
1681 }

```

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).

```

1682 \cs_set_eq:NN \@@_old_ldots: \ldots
1683 \cs_set_eq:NN \@@_old_cdots: \cdots
1684 \cs_set_eq:NN \@@_old_vdots: \vdots
1685 \cs_set_eq:NN \@@_old_ddots: \ddots
1686 \cs_set_eq:NN \@@_old_iddots: \iddots
1687 \bool_if:NTF \l_@@_standard_cline_bool
1688 { \cs_set_eq:NN \cline \@@_standard_cline: }
1689 { \cs_set_eq:NN \cline \@@_cline: }
1690 \cs_set_eq:NN \Ldots \@@_Ldots:
1691 \cs_set_eq:NN \Cdots \@@_Cdots:
1692 \cs_set_eq:NN \Vdots \@@_Vdots:
1693 \cs_set_eq:NN \Ddots \@@_Ddots:
1694 \cs_set_eq:NN \Iddots \@@_Iddots:
1695 \cs_set_eq:NN \Hline \@@_Hline:
1696 \cs_set_eq:NN \Hspace \@@_Hspace:
1697 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1698 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1699 \cs_set_eq:NN \Block \@@_Block:
1700 \cs_set_eq:NN \rotate \@@_rotate:
1701 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1702 \cs_set_eq:NN \dotfill \@@_dotfill:
1703 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1704 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1705 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1706 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1707 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1708 \cs_set_eq:NN \TopRule \@@_TopRule
1709 \cs_set_eq:NN \MidRule \@@_MidRule
1710 \cs_set_eq:NN \BottomRule \@@_BottomRule
1711 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1712 \cs_set_eq:NN \Hbrace \@@_Hbrace
1713 \cs_set_eq:NN \Vbrace \@@_Vbrace
1714 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1715 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1716 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1717 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1718 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1719 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1720 \int_if_zero:nTF \l_@@_first_row_int
1721 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1722 {
1723   \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1724   { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }

```

```

1725     }
1726     \bool_if:NT \l_@@_renew_dots_bool { \@@_renew_dots: }

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1727     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1728     \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1729         { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1730     \@@_revert_colortbl:

```

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

```

1731     \tl_if_exist:NT \l_@@_note_in_caption_tl
1732     {
1733         \tl_if_empty:NF \l_@@_note_in_caption_tl
1734         {
1735             \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1736             \int_gset:Nn \c@tablarnote \l_@@_note_in_caption_tl
1737         }
1738     }

```

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).

```

1739     \seq_gclear:N \g_@@_multicolumn_cells_seq
1740     \seq_gclear:N \g_@@_multicolumn_sizes_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).

```

1741     \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`).

```

1742     \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.

```

1743     \int_gzero:N \g_@@_col_total_int
1744     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1745     \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.

```

1746     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1747     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1748     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1749     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1750     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1751     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1752     \tl_gclear:N \g_nicematrix_code_before_tl
1753     \tl_gclear:N \g_@@_pre_code_before_tl

```

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

```

1754 \dim_zero_new:N \l_@@_left_delim_dim
1755 \dim_zero_new:N \l_@@_right_delim_dim
1756 \bool_if:NTF \g_@@_delims_bool
1757 {

```

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

```

1758 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1759 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1760 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1761 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1762 }
1763 {
1764 \dim_gset:Nn \l_@@_left_delim_dim
1765 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1766 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1767 }
1768 }

```

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.

```

1769 \cs_new_protected:Npn \@@_pre_array:
1770 {
1771 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1772 \int_gzero_new:N \c@iRow
1773 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1774 \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).

```

1775 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1776 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1777 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1778 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1779 \bool_if:NT \g_@@_aux_found_bool
1780 {
1781 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1782 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1783 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1784 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1785 }

```

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).

```

1786 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1787 {
1788 \bool_set_true:N \l_@@_last_row_without_value_bool
1789 \bool_if:NT \g_@@_aux_found_bool
1790 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1791 }
1792 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1793 {
1794 \bool_if:NT \g_@@_aux_found_bool

```

```

1795     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1796   }

```

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”.

```

1797   \int_compare:nNtT \l_@@_last_row_int > { -2 }
1798   {
1799     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1800     {
1801       \dim_compare:nNtT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1802       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1803       \dim_compare:nNtT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1804       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1805     }
1806   }

1807   \seq_gclear:N \g_@@_cols_vlism_seq
1808   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1809   \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1810   \@@_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).

```

1811   \hbox_set:Nw \l_@@_the_array_box
1812   \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1813   \skip_horizontal:N \l_@@_extra_left_margin_dim
1814   \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).

```

1815   \m@th
1816   $ % $
1817   \bool_if:NTF \l_@@_light_syntax_bool
1818   { \use:c { @@-light-syntax } }
1819   { \use:c { @@-normal-syntax } }
1820   }

```

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

```

1821   \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1822   {
1823     \tl_set:Nn \l_tmpa_tl { #1 }
1824     \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1825     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1826     \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1827     \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...

```

1828   \@@_pre_array:
1829   }

```

## 9 The \CodeBefore

```

1830 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1831 \cs_new_protected_nopar:Npn \@@_CodeBefore: { \@@_fatal:n { Bad~use~of~CodeBefore } }

```

The following command will be executed if the \CodeBefore has to be actually executed (that command will be used only once and is present alone only for legibility).

```

1832 \cs_new_protected:Npn \@@_pre_code_before:
1833 {

```

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.

```

1834 \pgfsys@markposition { \@@_env: - position }
1835 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1836 \pgfpicture
1837 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1838 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1839 {
1840 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1841 \pgfcoordinate { \@@_env: - row - ##1 }
1842 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1843 }

```

Now, the recreation of the col nodes.

```

1844 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1845 {
1846 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1847 \pgfcoordinate { \@@_env: - col - ##1 }
1848 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1849 }

```

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

```

1850 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1851 \endpgfpicture

```

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

```

1852 \@@_create_diag_nodes:

```

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

```

1853 \@@_create_blocks_nodes:
1854 \IfPackageLoadedT { tikz }
1855 {
1856 \tikzset
1857 {
1858 every-picture / .style =
1859 { overlay , name~prefix = \@@_env: - }
1860 }
1861 }
1862 \cs_set_eq:NN \cellcolor \@@_cellcolor
1863 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1864 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1865 \cs_set_eq:NN \rowcolor \@@_rowcolor
1866 \cs_set_eq:NN \rowcolors \@@_rowcolors
1867 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1868 \cs_set_eq:NN \arraycolor \@@_arraycolor
1869 \cs_set_eq:NN \columncolor \@@_columncolor
1870 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1871 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1872 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1873 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1874 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1875 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1876 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1877 }

```

```

1878 \cs_new_protected:Npn \@@_exec_code_before:
1879 {

```

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...

```

1880 \clist_map_inline:Nn \l_@@_corners_cells_clist
1881 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1882 \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`.

```

1883 \@@_add_to_colors_seq:nn { { nocolor } } { }
1884 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1885 \group_begin:

```

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

```

1886 \if_mode_math:
1887 \@@_exec_code_before_i:
1888 \else:
1889 $ % $
1890 \@@_exec_code_before_i:
1891 $ % $
1892 \fi:
1893 \group_end:
1894 }

```

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`).

```

1895 \cs_new_protected:Npn \@@_exec_code_before_i:
1896 {
1897 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1898 { \@@_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`.

```

1899 \exp_last_unbraced:No \@@_CodeBefore_keys:
1900 \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1901 \@@_actually_color:
1902 \l_@@_code_before_tl
1903 \q_stop
1904 }

```

```

1905 \keys_define:nn { nicematrix / CodeBefore }
1906 {
1907 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1908 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1909 sub-matrix .value_required:n = true ,
1910 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1911 delimiters / color .value_required:n = true ,
1912 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1913 }

```

```

1914 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1915 {
1916   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1917   \@@_CodeBefore:w
1918 }

```

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.

```

1919 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1920 {
1921   \bool_if:NTF \g_@@_aux_found_bool
1922   {
1923     \@@_pre_code_before:
1924     \legacy_if:nF { measuring@ } { #1 }
1925   }

```

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).

```

1926 {
1927   \pgfsys@markposition { \@@_env: - position }
1928   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1929   \pgfpicture
1930   \pgf@relevantforpicturesizefalse
1931   \endpgfpicture

```

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

```

1932   \pgfpicture
1933   \pgfrememberpicturepositiononpagetrue
1934   \endpgfpicture

```

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

```

1935   \pgfpicture
1936   \pgf@relevantforpicturesizefalse
1937   \pgfrememberpicturepositiononpagetrue
1938   \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1939   \pgfpicture
1940   \pgf@relevantforpicturesizefalse
1941   \endpgfpicture
1942 }
1943 }

```

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.

```

1944 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1945 {
1946   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1947   {
1948     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1949     \pgfcoordinate { \@@_env: - row - ##1 - base }
1950     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1951     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1952     {
1953       \cs_if_exist:cT
1954       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1955       {
1956         \pgfsys@getposition
1957         { \@@_env: - ##1 - #####1 - NW }
1958         \@@_node_position:
1959         \pgfsys@getposition

```

```

1960         { \@@_env: - ##1 - ####1 - SE }
1961         \@@_node_position_i:
1962         \@@_pgf_rect_node:nnn
1963         { \@@_env: - ##1 - ####1 }
1964         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1965         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1966     }
1967 }
1968 }
1969 \@@_create_extra_nodes:
1970 \@@_create_aliases_last:
1971 }

1972 \cs_new_protected:Npn \@@_create_aliases_last:
1973 {
1974     \int_step_inline:nn \c@iRow
1975     {
1976         \pgfnodealias
1977         { \@@_env: - ##1 - last }
1978         { \@@_env: - ##1 - \int_use:N \c@jCol }
1979     }
1980     \int_step_inline:nn \c@jCol
1981     {
1982         \pgfnodealias
1983         { \@@_env: - last - ##1 }
1984         { \@@_env: - \int_use:N \c@iRow - ##1 }
1985     }
1986     \pgfnodealias
1987     { \@@_env: - last - last }
1988     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1989 }

1990 \cs_new_protected:Npn \@@_create_blocks_nodes:
1991 {
1992     \pgfpicture
1993     \pgf@relevantforpicturesizefalse
1994     \pgfrememberpicturepositiononpagetrue
1995     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1996     { \@@_create_one_block_node:nnnnn ##1 }
1997     \endpgfpicture
1998 }

```

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

```

1999 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2000 {
2001     \tl_if_empty:nF { #5 }
2002     {
2003         \@@_qpoint:n { col - #2 }
2004         \dim_set_eq:NN \l_tmpa_dim \pgf@x
2005         \@@_qpoint:n { #1 }
2006         \dim_set_eq:NN \l_tmpb_dim \pgf@y
2007         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2008         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2009         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2010         % \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y % 2026-02-24
2011         \@@_pgf_rect_node:nnnnn
2012         { \@@_env: - #5 }
2013         { \dim_use:N \l_tmpa_dim }

```

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

```

2014         { \dim_use:N \l_tmpb_dim }
2015         { \dim_use:N \l_@@_tmpc_dim }
2016         { \dim_use:N \pgf@y }
2017     }
2018 }

```

## 10 The environment `{NiceArrayWithDelims}`

```

2019 \NewDocumentEnvironment { NiceArrayWithDelims }
2020 { m m 0 { } m ! 0 { } t \CodeBefore }
2021 {

```

```

2022     \@@_provide_pgfsyspdfmark:
2023     \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.

```

2024     \bgroup

2025     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2026     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2027     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2028     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2029     \int_gzero:N \g_@@_block_box_int
2030     \dim_gzero:N \g_@@_width_last_col_dim
2031     \dim_gzero:N \g_@@_width_first_col_dim
2032     \bool_gset_false:N \g_@@_row_of_col_done_bool
2033     \str_if_empty:NT \g_@@_name_env_str
2034     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2035     \bool_if:NTF \l_@@_tabular_bool
2036     \mode_leave_vertical:
2037     \@@_test_if_math_mode:
2038     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2039     \bool_set_true:N \l_@@_in_env_bool

```

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

```

2040     \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.

```

2041     \cs_if_exist:NT \tikz@library@external@loaded
2042     {
2043         \tikzexternaldisable
2044         \cs_if_exist:NT \ifstandalone
2045         { \tikzset { external / optimize = false } }
2046     }

```

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

```

2047     \int_gincr:N \g_@@_env_int
2048     \bool_if:NF \l_@@_block_auto_columns_width_bool
2049     { \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.

```

2050     \seq_gclear:N \g_@@_blocks_seq

```

---

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

```
2051 \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`.

```
2052 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
```

```
2053 \seq_gclear:N \g_@@_pos_of_xdots_seq
```

```
2054 \tl_gclear_new:N \g_@@_code_before_tl
```

```
2055 \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.

```
2056 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl } {
```

```
2057 {
```

```
2058 \bool_gset_true:N \g_@@_aux_found_bool
```

```
2059 \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl } {
```

```
2060 }
```

```
2061 { \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.

```
2062 \tl_gclear:N \g_@@_aux_tl
```

```
2063 \tl_if_empty:NF \g_@@_code_before_tl {
```

```
2064 {
```

```
2065 \bool_set_true:N \l_@@_code_before_bool
```

```
2066 \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
```

```
2067 }
```

```
2068 \tl_if_empty:NF \g_@@_pre_code_before_tl { \bool_set_true:N \l_@@_code_before_bool }
```

```
2069 }
```

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`.

```
2070 \bool_if:NTF \g_@@_delims_bool {
```

```
2071 { \keys_set:nn { nicematrix / pNiceArray } }
```

```
2072 { \keys_set:nn { nicematrix / NiceArray } }
```

```
2073 { #3 , #5 }
```

```
2074 \@@_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:`.

```
2075 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
```

```
2076 }
```

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

```
2077 {
```

```
2078 \bool_if:NTF \l_@@_light_syntax_bool {
```

```
2079 { \use:c { end @@-light-syntax } }
```

```
2080 { \use:c { end @@-normal-syntax } }
```

```
2081 $ % $
```

```
2082 \skip_horizontal:N \l_@@_right_margin_dim
```

```
2083 \skip_horizontal:N \l_@@_extra_right_margin_dim
```

```
2084 \hbox_set_end:
```

```
2085 \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.

```
2086 \bool_if:NT \l_@@_width_used_bool {
```

```
2087 }
```

```

2088 \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2089 { \@@_error_or_warning:n { width-without-X-columns } }
2090 }

```

Now, if there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a X-column of weight  $x$ , the width will be `l_@@_X_columns_dim` multiplied by  $x$ .

```

2091 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2092 \@@_compute_width_X:

```

It 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).

```

2093 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2094 {
2095   \bool_if:NF \l_@@_last_row_without_value_bool
2096   {
2097     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2098     {
2099       \@@_error:n { Wrong-last-row }
2100       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2101     }
2102   }
2103 }

```

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

```

2104 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2105 \bool_if:NTF \g_@@_last_col_found_bool
2106 { \int_gdecr:N \c@jCol }
2107 {
2108   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2109   { \@@_error:n { last-col-not-used } }
2110 }

```

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

```

2111 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2112 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2113 { \int_gdecr:N \c@iRow }

```

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

```

2114 \int_if_zero:nT \l_@@_first_col_int
2115 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2116 \bool_if:nTF { ! \g_@@_delims_bool }
2117 {
2118   \str_if_eq:eeTF c \l_@@_baseline_tl
2119   \@@_use_arraybox_with_notes_c:
2120   {
2121     \str_if_eq:eeTF b \l_@@_baseline_tl
2122     \@@_use_arraybox_with_notes_b:
2123     \@@_use_arraybox_with_notes:
2124   }
2125 }

```

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).

```

2126 {

```

---

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

```

2127 \int_if_zero:nTF \l_@@_first_row_int
2128 {
2129     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2130     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2131 }
2132 { \dim_zero:N \l_tmpa_dim }

```

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

```

2133 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2134 {
2135     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2136     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2137 }
2138 { \dim_zero:N \l_tmpb_dim }
2139 \hbox_set:Nn \l_tmpa_box
2140 {
2141     \m@th
2142     $ % $
2143     \@@_color:o \l_@@_delimiters_color_tl
2144     \exp_after:wN \left \g_@@_left_delim_tl
2145     \vcenter
2146     {

```

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.

```

2147         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2148         \hbox
2149         {
2150             \bool_if:NTF \l_@@_tabular_bool
2151             { \skip_horizontal:n { - \tabcolsep } }
2152             { \skip_horizontal:n { - \arraycolsep } }
2153             \@@_use_arraybox_with_notes_c:
2154             \bool_if:NTF \l_@@_tabular_bool
2155             { \skip_horizontal:n { - \tabcolsep } }
2156             { \skip_horizontal:n { - \arraycolsep } }
2157         }

```

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

```

2158         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2159     }
2160     \exp_after:wN \right \g_@@_right_delim_tl
2161     $ % $
2162 }

```

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.

```

2163     \bool_if:NTF \l_@@_delimiters_max_width_bool
2164     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2165     \@@_put_box_in_flow:
2166 }

```

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

```

2167 \bool_if:NT \g_@@_last_col_found_bool
2168 { \skip_horizontal:N \g_@@_width_last_col_dim }
2169 \bool_if:NT \l_@@_preamble_bool
2170 {
2171     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2172     { \@@_err_columns_not_used: }
2173 }

```

---

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

2174 \@@\_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.

2175 \egroup

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

```
2176 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2177 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2178 \iow_now:Ne \@mainaux
2179 {
2180   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2181   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2182   { \exp_not:o \g_@@_aux_tl }
2183 }
2184 \iow_now:Nn \@mainaux { \ExplSyntaxOff }
```

```
2185 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2186 }
```

This is the end of the environment {NiceArrayWithDelims}.

```
2187 \cs_new_protected:Npn \@@_err_columns_not_used:
2188 {
2189   \@@_warning:n { columns~not~used }
2190   \cs_gset:Npn \@@_err_columns_not_used: { }
2191 }
```

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  $x$ .

```
2192 \cs_new_protected:Npn \@@_compute_width_X:
2193 {
2194   \tl_gput_right:Ne \g_@@_aux_tl
2195   {
2196     \bool_set_true:N \l_@@_X_columns_aux_bool
2197     \dim_set:Nn \l_@@_X_columns_dim
2198     {
```

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

```
2199     \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2200     { \dim_use:N \l_@@_X_columns_dim }
2201     {
2202       \dim_compare:nNnTF
2203       {
2204         \dim_abs:n
2205         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2206       }
2207       <
2208       { 0.001 pt }
2209       { \dim_use:N \l_@@_X_columns_dim }
2210       {
2211         \dim_eval:n
2212         {
2213           \l_@@_X_columns_dim
2214           +
2215           \fp_to_dim:n
2216           {
2217             (
2218               \dim_eval:n
```

```

2219             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2220         )
2221         / \fp_use:N \g_@@_total_X_weight_fp
2222     }
2223 }
2224 }
2225 }
2226 }
2227 }
2228 }

```

## 11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

The preamble given by the final user is stored in `\g_@@_user_preamble_tl`. The modified version will be stored in `\g_@@_array_preamble_tl`.

```

2229 \cs_new_protected:Npn \@@_transform_preamble:
2230 {
2231     \@@_transform_preamble_i:
2232     \@@_transform_preamble_ii:
2233 }
2234 \cs_new_protected:Npn \@@_transform_preamble_i:
2235 {
2236     \int_gzero:N \c@jCol

```

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

```

2237     \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2238     \bool_gset_false:N \g_tmpb_bool

```

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

```

2239     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2240     \int_zero:N \l_tmpa_int
2241     \tl_gclear:N \g_@@_array_preamble_tl
2242     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2243     {
2244         \tl_gset:Nn \g_@@_array_preamble_tl
2245         { ! { \skip_horizontal:N \arrayrulewidth } }
2246     }
2247     {
2248         \clist_if_in:NnT \l_@@_vlines_clist 1
2249         {
2250             \tl_gset:Nn \g_@@_array_preamble_tl
2251             { ! { \skip_horizontal:N \arrayrulewidth } }
2252         }
2253     }

```

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

```

2254     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2255     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2256 \@@_replace_columncolor:
2257 }

```

```

2258 \cs_new_protected:Npn \@@_transform_preamble_ii:
2259 {

```

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

```

2260 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2261 {
2262     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2263     { \bool_gset_true:N \g_@@_delims_bool }
2264 }
2265 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2266 \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

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

```

2267 \int_if_zero:nTF \l_@@_first_col_int
2268 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2269 {
2270     \bool_if:NF \g_@@_delims_bool
2271     {
2272         \bool_if:NF \l_@@_tabular_bool
2273         {
2274             \clist_if_empty:NT \l_@@_vlines_clist
2275             {
2276                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2277                 { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2278             }
2279         }
2280     }
2281 }
2282 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2283 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2284 {
2285     \bool_if:NF \g_@@_delims_bool
2286     {
2287         \bool_if:NF \l_@@_tabular_bool
2288         {
2289             \clist_if_empty:NT \l_@@_vlines_clist
2290             {
2291                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2292                 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2293             }
2294         }
2295     }
2296 }

```

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.

```

2297 \tag_if_active:F
2298 {

```

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

```

2299 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2300 {
2301     \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2302         { > { \@@_err_too_many_cols: } 1 }
2303     }
2304 }
2305 }

```

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.

```

2306 \cs_new_protected:Npn \@@_rec_preamble:n #1
2307 {

```

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

```

2308     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2309     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2310     {

```

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

```

2311     \cs_if_exist:cTF { NC @ find @ #1 }
2312     {
2313         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2314         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2315     }
2316     {
2317         \str_if_eq:nnTF { #1 } { S }
2318         { \@@_fatal:n { unknown~column~type~S } }
2319         { \@@_fatal:nn { unknown~column~type } { #1 } }
2320     }
2321 }
2322 }

```

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

```

2323 \cs_new_protected:Npn \@@_c: #1
2324 {
2325     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2326     \tl_gclear:N \g_@@_pre_cell_tl
2327     \tl_gput_right:Nn \g_@@_array_preamble_tl
2328     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2329     \int_gincr:N \c@jCol
2330     \@@_rec_preamble_after_col:n
2331 }
2332 \cs_new_protected:Npn \@@_l: #1
2333 {
2334     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2335     \tl_gclear:N \g_@@_pre_cell_tl
2336     \tl_gput_right:Nn \g_@@_array_preamble_tl
2337     {
2338         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2339         l
2340         < \@@_cell_end:
2341     }
2342     \int_gincr:N \c@jCol
2343     \@@_rec_preamble_after_col:n
2344 }

```

---

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

```

2345 \cs_new_protected:Npn \@@_r: #1
2346 {
2347   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2348   \tl_gclear:N \g_@@_pre_cell_tl
2349   \tl_gput_right:Nn \g_@@_array_preamble_tl
2350   {
2351     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2352     r
2353     < \@@_cell_end:
2354   }
2355   \int_gincr:N \c@jCol
2356   \@@_rec_preamble_after_col:n
2357 }

```

For ! and @

```

2358 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2359 {
2360   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2361   \@@_rec_preamble:n
2362 }
2363 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2364 \cs_new_protected:cpn { @@ _ | : } #1
2365 {
\l_tmpa_int is the number of successive occurrences of |
2366   \int_incr:N \l_tmpa_int
2367   \@@_make_preamble_i_i:n
2368 }
2369 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2370 {

```

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

```

2371   \str_if_eq:nnTF { #1 } { | }
2372   { \use:c { @@ _ | : } | }
2373   { \@@_make_preamble_i_ii:nn { } #1 }
2374 }

```

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

```

2375 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2376 {
2377   \str_if_eq:nnTF { #2 } { [ ]
2378   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2379   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2380 }
2381 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2382 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2383 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2384 {
2385   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2386   \tl_gput_right:Ne \g_@@_array_preamble_tl
2387   {

```

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

```

2388   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2389 }
2390 \tl_gput_right:Ne \g_@@_rules_tl
2391 {

```

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

```

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

```

We will use a version slightly more efficient:

```

2392 {
2393   \int_compare:nNt \l_tmpa_int > 1
2394   { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2395   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2396   \dim_set:Nn \l_@@_rule_width_after_dim
2397   { \dim_use:N \l_@@_rule_width_before_dim }
2398   \@@_draw_vrule:n { #2 }
2399 }

```

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.

```

2400 }
2401 \int_zero:N \l_tmpa_int
2402 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2403 \@@_rec_preamble:n #1
2404 }

2405 \cs_new_protected:cpn { @@_ > : } #1 #2
2406 {
2407   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2408   \@@_rec_preamble:n
2409 }

2410 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier `p` (and also the specifiers `m`, `b`, `V` and `X`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys.

```

2411 \keys_define:nn { nicematrix / p-column }
2412 {
2413   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2414   r .value_forbidden:n = true ,
2415   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2416   c .value_forbidden:n = true ,
2417   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2418   l .value_forbidden:n = true ,
2419   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2420   S .value_forbidden:n = true ,
2421   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2422   p .value_forbidden:n = true ,
2423   t .meta:n = p ,
2424   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2425   m .value_forbidden:n = true ,
2426   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2427   b .value_forbidden:n = true
2428 }

```

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

```

2429 \cs_new_protected:Npn \@@_p: #1
2430 {
2431   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2432 \@@_make_preamble_ii_i:n
2433 }
2434 \cs_new_eq:NN \@@_b: \@@_p:
2435 \cs_new_eq:NN \@@_m: \@@_p:
2436 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2437 {
2438   \str_if_eq:nnTF { #1 } { [ ] }
2439   { \@@_make_preamble_ii_ii:w [ ] }
2440   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2441 }
2442 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2443 { \@@_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.

```

2444 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2445 {

```

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).

```

2446 \str_set:Nn \l_@@_hpos_col_str { j }
2447 \@@_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`.

```

2448 \setlength { \l_tmpa_dim } { #2 }
2449 \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2450 }
2451 \cs_new_protected:Npn \@@_keys_p_column:n #1
2452 { \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.

```

2453 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2454 {

```

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

```

2455 \use:e
2456 {
2457   \@@_make_preamble_ii_vi:nnnnnnnn
2458   { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2459   { #1 }
2460   {
2461     \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.

```

2462 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2463 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2464 {

```

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

```

2465         \def \exp_not:N \l_@@_hpos_cell_tl
2466         { \str_lowercase:f { \l_@@_hpos_col_str } }
2467     }
2468     \IfPackageLoadedTF { ragged2e }
2469     {
2470         \str_case:on \l_@@_hpos_col_str
2471         {

```

The following `\exp_not:N` are mandatory.

```

2472         c { \exp_not:N \Centering }
2473         l { \exp_not:N \RaggedRight }
2474         r { \exp_not:N \RaggedLeft }
2475     }
2476 }
2477 {
2478     \str_case:on \l_@@_hpos_col_str
2479     {
2480         c { \exp_not:N \centering }
2481         l { \exp_not:N \raggedright }
2482         r { \exp_not:N \raggedleft }
2483     }
2484 }
2485 #3
2486 }
2487 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2488 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2489 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2490 { #2 }
2491 {
2492     \str_case:onF \l_@@_hpos_col_str
2493     {
2494         { j } { c }
2495         { si } { c }
2496     }

```

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

```

2497         { \str_lowercase:f \l_@@_hpos_col_str }
2498     }
2499 }

```

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

```

2500     \int_gincr:N \c@jCol
2501     \@@_rec_preamble_after_col:n
2502 }

```

**#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*.

```

2503 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2504 {
2505     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2506     {

```

```

2507     \tl_gput_right:Nn \g_@@_array_preamble_tl
2508     { > \@@_test_if_empty_for_S: }
2509   }
2510   {
2511     \str_if_eq:eeTF { #7 } { varwidth }
2512     {
2513       \tl_gput_right:Nn \g_@@_array_preamble_tl
2514       { > \@@_test_if_empty_varwidth: }
2515     }
2516     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2517   }
2518   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2519   \tl_gclear:N \g_@@_pre_cell_tl
2520   \tl_gput_right:Nn \g_@@_array_preamble_tl
2521   {
2522     > {

```

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.

```

2523     \dim_set:Nn \l_@@_col_width_dim { #2 }
2524     \@@_cell_begin:

```

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

```

2525     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2526     \everypar
2527     {
2528       \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2529       \everypar { }
2530     }

```

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

```

2531     #3

```

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

```

2532     \g_@@_row_style_tl
2533     \arraybackslash
2534     #5
2535   }
2536   #8
2537   < {
2538     #6

```

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

```

2539     \@finalstrut \@arstrutbox
2540     \use:c { end #7 }

```

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

```

2541     #4
2542     \@@_cell_end:
2543   }
2544 }
2545 }

```

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

```

2546 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2547 {

```

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 `&`).

```

2548 \group_align_safe_begin:
2549 \peek_meaning:NTF &
2550 \@@_the_cell_is_empty:
2551 {
2552   \peek_meaning:NTF \\\
2553   \@@_the_cell_is_empty:
2554   {
2555     \peek_meaning:NTF \crcr
2556     \@@_the_cell_is_empty:
2557     \group_align_safe_end:
2558   }
2559 }
2560 }

```

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

```

2561 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2562 {
2563   \group_align_safe_begin:
2564   \peek_meaning:NTF &
2565   \@@_the_cell_is_empty_varwidth:
2566   {
2567     \peek_meaning:NTF \\\
2568     \@@_the_cell_is_empty_varwidth:
2569     {
2570       \peek_meaning:NTF \crcr
2571       \@@_the_cell_is_empty_varwidth:
2572       \group_align_safe_end:
2573     }
2574   }
2575 }

2576 \cs_new_protected:Npn \@@_the_cell_is_empty:
2577 {
2578   \group_align_safe_end:
2579   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2580   {

```

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

```

2581 \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).

```

2582 \skip_horizontal:N \l_@@_col_width_dim
2583 }
2584 }

2585 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2586 {
2587   \group_align_safe_end:
2588   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2589   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2590 }

2591 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2592 {
2593   \peek_meaning:NT \__siunitx_table_skip:n
2594   { \bool_gset_true:N \g_@@_empty_cell_bool }
2595 }

```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the

cell. However, we consider (as in `array`) that if the height of the cell is no more than the height of `\strutbox`, there is only one row.

```
2596 \cs_new_protected:Npn \l_@@_center_cell_box:
2597 {
```

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

```
2598 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2599 {
2600   \dim_compare:nNnT
2601     { \box_ht:N \l_@@_cell_box }
2602     >
```

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).

```
2603   { \box_ht:N \strutbox }
2604   {
2605     \hbox_set:Nn \l_@@_cell_box
2606       {
2607         \box_move_down:nn
2608           {
2609             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2610               + \baselineskip ) / 2
2611           }
2612         { \box_use:N \l_@@_cell_box }
2613       }
2614     }
2615   }
2616 }
```

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

```
2617 \cs_new_protected:Npn \l_@@_V: #1 #2
2618 {
2619   \str_if_eq:nnTF { #2 } { [ ] }
2620     { \l_@@_make_preamble_V_i:w [ ] }
2621     { \l_@@_make_preamble_V_i:w [ ] { #2 } }
2622 }
2623 \cs_new_protected:Npn \l_@@_make_preamble_V_i:w [ #1 ]
2624 { \l_@@_make_preamble_V_ii:nn { #1 } }
2625 \cs_new_protected:Npn \l_@@_make_preamble_V_ii:nn #1 #2
2626 {
2627   \str_set:Nn \l_@@_vpos_col_str { p }
2628   \str_set:Nn \l_@@_hpos_col_str { j }
2629   \l_@@_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`.

```
2630   \setlength { \l_tmpa_dim } { #2 }
2631   \IfPackageLoadedTF { varwidth }
2632     { \l_@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2633     {
2634       \l_@@_error_or_warning:n { varwidth-not-loaded }
2635       \l_@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2636     }
2637 }
2638 % \end{macrocode}
2639 %
2640 % \medskip
2641 % For |w| and |W|
2642 % \begin{macrocode}
2643 \cs_new_protected:Npn \l_@@_w: { \l_@@_make_preamble_w:nnnn { } }
```

```

2644 \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.

2645 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2646 {
2647   \tl_if_in:nnTF { clr } { #3 }
2648   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2649   {
2650     \@@_error:nn { Invalid-argument-for~w } { #3 }
2651     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2652   }
2653 }

2654 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2655 {
2656   \str_if_eq:nnTF { #3 } { s }
2657   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2658   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2659 }

```

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.

```

2660 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2661 {
2662   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2663   \tl_gclear:N \g_@@_pre_cell_tl
2664   \tl_gput_right:Nn \g_@@_array_preamble_tl
2665   {
2666     > {

```

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.

```

2667       \setlength { \l_@@_col_width_dim } { #2 }
2668       \@@_cell_begin:
2669       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2670     }
2671     c
2672     < {
2673       \@@_cell_end_for_w_s:
2674       #1
2675       \@@_adjust_size_box:
2676       \box_use_drop:N \l_@@_cell_box
2677     }
2678   }
2679   \int_gincr:N \c@jCol
2680   \@@_rec_preamble_after_col:n
2681 }

```

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

```

2682 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2683 {
2684   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2685   \tl_gclear:N \g_@@_pre_cell_tl
2686   \tl_gput_right:Nn \g_@@_array_preamble_tl
2687   {
2688     > {

```

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

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

```

2689         \setlength { \l_@@_col_width_dim } { #4 }
2690         \hbox_set:Nw \l_@@_cell_box
2691         \@@_cell_begin:
2692         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2693     }
2694     c
2695     < {
2696         \@@_cell_end:
2697         \hbox_set_end:
2698         #1
2699         \@@_adjust_size_box:
2700         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2701     }
2702 }
```

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

```

2703     \int_gincr:N \c@jCol
2704     \@@_rec_preamble_after_col:n
2705 }

2706 \cs_new_protected:Npn \@@_special_W:
2707 {
2708     \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2709     { \@@_warning:n { W~warning } }
2710 }
```

For `S` (of `siunitx`).

```

2711 \cs_new_protected:Npn \@@_S: #1 #2
2712 {
2713     \str_if_eq:nnTF { #2 } { [ ] }
2714     { \@@_make_preamble_S:w [ ] }
2715     { \@@_make_preamble_S:w [ ] { #2 } }
2716 }

2717 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2718 { \@@_make_preamble_S:i:n { #1 } }

2719 \cs_new_protected:Npn \@@_test_siunitx:
2720 {
2721     \IfPackageLoadedTF
2722     { siunitx }
2723     {
2724         \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2725         { \@@_fatal:n { siunitx~too~old } }
2726     }
2727     { \@@_fatal:n { siunitx~not~loaded } }
2728     \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2729 }
2730 % \end{macrocode}
2731 %
2732 % \begin{macrocode}
2733 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2734 {
2735     \@@_test_siunitx:
2736     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2737     \tl_gclear:N \g_@@_pre_cell_tl
2738     \tl_gput_right:Nn \g_@@_array_preamble_tl
2739     {
2740         > {
```

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).

```

2741         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2742         \keys_set:nn { siunitx } { #1 }
2743         \@@_cell_begin:
2744         \siunitx_cell_begin:w
2745     }
2746     c
2747     <
2748     {
2749         \siunitx_cell_end:

```

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

```

2750         \tl_gput_right:Nc \g_@@_cell_after_hook_tl
2751         {
2752             \bool_if:NTF \l__siunitx_table_text_bool
2753             \bool_set_true:N
2754             \bool_set_false:N
2755             \l__siunitx_table_text_bool
2756         }
2757         \@@_cell_end:
2758     }
2759 }

```

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

```

2760     \int_gincr:N \c@jCol
2761     \@@_rec_preamble_after_col:n
2762 }

```

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

```

2763 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2764 {
2765     \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.

```

2766     \int_if_zero:nTF \c@jCol
2767     {
2768         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2769         {

```

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

```

2770         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2771         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2772         \@@_rec_preamble:n #2
2773     }
2774     {
2775         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2776         \@@_make_preamble_iv:nn { #1 } { #2 }
2777     }
2778 }
2779 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2780 }
2781 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2782 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2783 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2784 {
2785     \tl_gput_right:Nc \g_@@_pre_code_after_tl
2786     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2787 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2788 {
2789   \@@_error:nn { delimiter~after~opening } { #2 }
2790   \@@_rec_preamble:n
2791 }
2792 { \@@_rec_preamble:n #2 }
2793 }

```

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

```

2794 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2795 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2796 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2797 {
2798   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2799   \tl_if_in:nnTF { ) ] \} } { #2 }
2800   { \@@_make_preamble_v:nnn #1 #2 }
2801   {
2802     \str_if_eq:nnTF \s_stop { #2 }
2803     {
2804       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2805       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2806       {
2807         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2808         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2809         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2810         \@@_rec_preamble:n #2
2811       }
2812     }
2813     {
2814       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2815       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2816       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2817       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2818       \@@_rec_preamble:n #2
2819     }
2820   }
2821 }
2822 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2823 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2824 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2825 {
2826   \str_if_eq:nnTF \s_stop { #3 }
2827   {
2828     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2829     {
2830       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2831       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2832       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2833       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2834     }
2835     {
2836       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2837       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2838       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2839       \@@_error:nn { double~closing~delimiter } { #2 }
2840     }
2841   }
2842   {

```

```

2843 \tl_gput_right:N\g_@@_pre_code_after_tl
2844 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2845 \@@_error:nn { double~closing~delimiter } { #2 }
2846 \@@_rec_preamble:n #3
2847 }
2848 }

2849 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2850 { \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 @{...}.

```

2851 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2852 {
2853   \str_if_eq:nnTF { #1 } { < }
2854   { \@@_rec_preamble_after_col_i:n }
2855   {
2856     \str_if_eq:nnTF { #1 } { @ }
2857     { \@@_rec_preamble_after_col_ii:n }
2858     {
2859       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2860       {
2861         \tl_gput_right:Nn \g_@@_array_preamble_tl
2862         { ! { \skip_horizontal:N \arrayrulewidth } }
2863       }
2864       {
2865         \clist_if_in:N\l_@@_vlines_clist
2866         { \int_eval:n { \c@jCol + 1 } }
2867         {
2868           \tl_gput_right:Nn \g_@@_array_preamble_tl
2869           { ! { \skip_horizontal:N \arrayrulewidth } }
2870         }
2871       }
2872       \@@_rec_preamble:n { #1 }
2873     }
2874   }
2875 }

2876 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2877 {
2878   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2879   \@@_rec_preamble_after_col:n
2880 }

```

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.

```

2881 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2882 {
2883   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2884   {
2885     \tl_gput_right:Nn \g_@@_array_preamble_tl
2886     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2887   }
2888   {
2889     \clist_if_in:N\l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2890     {
2891       \tl_gput_right:Nn \g_@@_array_preamble_tl
2892       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2893     }
2894     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2895   }
2896   \@@_rec_preamble:n
2897 }

```

```

2898 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2899 {
2900   \tl_clear:N \l_tmpa_tl
2901   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2902   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2903 }

```

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.

```

2904 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2905 { \@@_rec_preamble:n }

```

For the case of a letter `X`. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter `X`.

```

2906 \cs_new_protected:Npn \@@_X: #1 #2
2907 {
2908   \str_if_eq:nnTF { #2 } { [ ]
2909     { \@@_make_preamble_X:w [ ] }
2910     { \@@_make_preamble_X:w [ ] #2 }
2911   }
2912   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2913   { \@@_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`.

```

2914 \keys_define:nn { nicematrix / X-column }
2915 {
2916   V .code:n =
2917     \IfPackageLoadedTF { varwidth }
2918     {
2919       \bool_set_true:N \l_@@_V_of_X_bool
2920       \bool_gset_true:N \g_@@_V_of_X_bool
2921     }
2922     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2923   unknown .code:n =
2924     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2925     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2926     { \@@_error_or_warning:n { invalid~weight } }
2927 }

```

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

```

2928 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2929 {

```

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`).

```

2930   \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`).

```

2931   \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`).

```

2932   \fp_set:Nn \l_tmpa_fp { 1.0 }
2933   \@@_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.

```
2934 \bool_set_false:N \l_@@_V_of_X_bool
2935 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2936 \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).

```
2937 \bool_if:NTF \l_@@_X_columns_aux_bool
2938 {
2939 \@@_make_preamble_ii_iv:nnn
```

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

```
2940 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2941 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2942 { \@@_no_update_width: }
2943 }
```

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.

```
2944 {
2945 \tl_gput_right:Nn \g_@@_array_preamble_tl
2946 {
2947 > {
2948 \@@_cell_begin:
2949 \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`.

```
2950 \NotEmpty
```

The following code will nullify the box of the cell.

```
2951 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2952 { \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`.

```
2953 \begin { minipage } { 5 cm } \arraybackslash
2954 }
2955 c
2956 < {
2957 \end { minipage }
2958 \@@_cell_end:
2959 }
2960 }
2961 \int_gincr:N \c@jCol
2962 \@@_rec_preamble_after_col:n
2963 }
2964 }
```

```
2965 \cs_new_protected:Npn \@@_no_update_width:
2966 {
2967 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2968 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2969 }
```

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

```

2970 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2971 {
2972   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2973   { \int_eval:n { \c@jCol + 1 } }
2974   \tl_gput_right:Ne \g_@@_array_preamble_tl
2975   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2976   \@@_rec_preamble:n
2977 }

```

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.

```

2978 \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).

```

2979 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2980 { \@@_fatal:n { Preamble-forgotten } }
2981 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2982 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2983 { @@ _ \token_to_str:N \hline : }
2984 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2985 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2986 { @@ _ \token_to_str:N \hline : }
2987 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2988 { @@ _ \token_to_str:N \hline : }
2989 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2990 { @@ _ \token_to_str:N \hline : }
2991 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2992 { @@ _ \token_to_str:N \hline : }
2993 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
2994 { \@@_fatal:n { NiceTabularX~probably~required } }
2995 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
2996 { @@ _ \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).

```

2997 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2998 {

```

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`.

```

2999   \multispan { #1 }
3000   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3001   \begingroup
3002   \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3003   \tl_gclear:N \g_@@_preamble_tl
3004   \@@_make_m_preamble:n #2 \q_stop

```

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

```

3005   \def \@addamp
3006   {
3007     \legacy_if:nTF { @firstamp }
3008     { \legacy_if_set_false:n { @firstamp } }

```

```

3009         { \@preamerr 5 }
3010     }
3011     \exp_args:No \@mkpream \g_@@_preamble_tl
3012     \@addtopreamble \@empty
3013     \endgroup
3014     \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3015     \int_compare:nNtT { #1 } > 1
3016     {
3017         \seq_gput_right:Ne \g_@@_multicolumn_cells_seq          % left replaced by right
3018         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3019         \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 } % left replaced by right
3020         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3021         {
3022             {
3023                 \int_if_zero:nTF \c@jCol
3024                 { \int_eval:n { \c@iRow + 1 } }
3025                 { \int_use:N \c@iRow }
3026             }
3027             { \int_eval:n { \c@jCol + 1 } }
3028             {
3029                 \int_if_zero:nTF \c@jCol
3030                 { \int_eval:n { \c@iRow + 1 } }
3031                 { \int_use:N \c@iRow }
3032             }
3033             { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3034         { }
3035     }
3036 }

```

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

```

3037     \RenewDocumentCommand { \cellcolor } { 0 { } m }
3038     {
3039         \tl_gput_right:Ne \g_@@_pre_code_before_tl
3040         {
3041             \@@_rectanglecolor [ ##1 ]
3042             { \exp_not:n { ##2 } }
3043             { \int_use:N \c@iRow - \int_use:N \c@jCol }
3044             { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3045         }
3046         \ignorespaces
3047     }

```

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

```

3048     \def \@sharp { #3 }
3049     \@arstrut
3050     \@preamble
3051     \null

```

We add some lines.

```

3052     \int_gadd:Nn \c@jCol { #1 - 1 }
3053     \int_compare:nNtT \c@jCol > \g_@@_col_total_int
3054     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3055     \ignorespaces
3056 }

```

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`.

```

3057 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3058 {
3059   \str_case:nnF { #1 }
3060   {
3061     c { \@@_make_m_preamble_i:n #1 }
3062     l { \@@_make_m_preamble_i:n #1 }
3063     r { \@@_make_m_preamble_i:n #1 }
3064     > { \@@_make_m_preamble_ii:nn #1 }
3065     ! { \@@_make_m_preamble_ii:nn #1 }
3066     @ { \@@_make_m_preamble_ii:nn #1 }
3067     | { \@@_make_m_preamble_iii:n #1 }
3068     p { \@@_make_m_preamble_iv:nnn t #1 }
3069     m { \@@_make_m_preamble_iv:nnn c #1 }
3070     b { \@@_make_m_preamble_iv:nnn b #1 }
3071     w { \@@_make_m_preamble_v:nnnn { } #1 }
3072     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3073     \q_stop { }
3074   }
3075   {
3076     \cs_if_exist:cTF { NC @ find @ #1 }
3077     {
3078       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3079       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3080     }
3081     {
3082       \str_if_eq:nnTF { #1 } { S }
3083       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3084       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3085     }
3086   }
3087 }

```

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

```

3088 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3089 {
3090   \tl_gput_right:Nn \g_@@_preamble_tl
3091   {
3092     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3093     #1
3094     < \@@_cell_end:
3095   }

```

We test for the presence of a `<`.

```

3096   \@@_make_m_preamble_x:n
3097 }

```

For `>`, `!` and `@`

```

3098 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3099 {
3100   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3101   \@@_make_m_preamble:n
3102 }

```

For `|`

```

3103 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3104 {
3105   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3106   \@@_make_m_preamble:n
3107 }

```

For p, m and b

```

3108 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3109 {
3110   \tl_gput_right:Nn \g_@@_preamble_tl
3111   {
3112     > {
3113       \@@_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`.

```

3114       \setlength { \l_tmpa_dim } { #3 }
3115       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3116       \mode_leave_vertical:
3117       \arraybackslash
3118       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
3119     }
3120     c
3121     < {
3122       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
3123       \end { minipage }
3124       \@@_cell_end:
3125     }
3126   }

```

We test for the presence of a `<`.

```

3127   \@@_make_m_preamble_x:n
3128 }

```

For w and W

```

3129 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3130 {
3131   \tl_gput_right:Nn \g_@@_preamble_tl
3132   {
3133     > {
3134       \dim_set:Nn \l_@@_col_width_dim { #4 }
3135       \hbox_set:Nw \l_@@_cell_box
3136       \@@_cell_begin:
3137       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3138     }
3139     c
3140     < {
3141       \@@_cell_end:
3142       \hbox_set_end:
3143       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3144       #1
3145       \@@_adjust_size_box:
3146       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3147     }
3148   }

```

We test for the presence of a `<`.

```

3149   \@@_make_m_preamble_x:n
3150 }

```

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

```

3151 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3152 {
3153   \str_if_eq:nnTF { #1 } { < }
3154   \@@_make_m_preamble_ix:n
3155   { \@@_make_m_preamble:n { #1 } }
3156 }

```

```

3157 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3158 {
3159   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3160   \@@_make_m_preamble_x:n
3161 }

```

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`).

```

3162 \cs_new_protected:Npn \@@_put_box_in_flow:
3163 {
3164   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3165   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3166   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3167   { \box_use_drop:N \l_tmpa_box }
3168   \@@_put_box_in_flow_i:
3169 }

```

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

```

3170 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3171 {
3172   \pgfpicture
3173   \@@_qpoint:n { row - 1 }
3174   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3175   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3176   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3177   \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).

```

3178   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3179   {
3180     \int_set:Nn \l_tmpa_int
3181     { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3182     \bool_lazy_or:nnT
3183     { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3184     { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3185     {
3186       \@@_error:n { bad-value-for~baseline-line }
3187       \int_set:Nn \l_tmpa_int 1
3188     }
3189     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3190   }
3191   {
3192     \str_if_eq:eeTF t \l_@@_baseline_tl
3193     { \int_set:Nn \l_tmpa_int 1 }
3194     {
3195       \str_if_eq:eeTF b \l_@@_baseline_tl
3196       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3197       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3198     }
3199     \bool_lazy_or:nnT
3200     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3201     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3202     {
3203       \@@_error:n { bad-value-for~baseline }
3204       \int_set:Nn \l_tmpa_int 1
3205     }
3206     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3207         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3208     }
3209     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3210 \endpgfpicture
3211 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3212 \box_use_drop:N \l_tmpa_box
3213 }

```

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).

```

3214 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3215 {

```

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.

```

3216     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3217     {
3218         \int_compare:nNnT \c@jCol > 1
3219         {
3220             \box_set_wd:Nn \l_@@_the_array_box
3221             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3222         }
3223     }

```

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

```

3224 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3225 \bool_if:NT \l_@@_caption_above_bool
3226 {
3227     \tl_if_empty:NF \l_@@_caption_tl
3228     {
3229         \bool_set_false:N \g_@@_caption_finished_bool
3230         \int_gzero:N \c@tabularnote
3231         \@@_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).

```

3232         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3233         {
3234             \tl_gput_right:Ne \g_@@_aux_tl
3235             {
3236                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3237                 { \int_use:N \g_@@_notes_caption_int }
3238             }
3239             \int_gzero:N \g_@@_notes_caption_int
3240         }
3241     }
3242 }

```

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

```

3243 \hbox
3244 {
3245     \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.

```

3246     \@@_create_extra_nodes:
3247     \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3248 }

```

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).

```

3249     \bool_lazy_any:nT
3250     {
3251       { ! \seq_if_empty_p:N \g_@@_notes_seq }
3252       { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3253       { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3254     }
3255     {
3256       \bool_if:NTF \l_@@_notes_no_print_bool
3257       { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3258       \@@_tabular_notes:
3259     }
3260     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3261     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3262   \end { minipage }
3263 }

```

```

3264 \cs_new_protected:Npn \@@_insert_caption:
3265 {
3266   \tl_if_empty:NF \l_@@_caption_tl
3267   {
3268     \cs_if_exist:NTF \@capytype
3269     { \@@_insert_caption_i: }
3270     { \@@_error:n { caption~outside~float } }
3271   }
3272 }

```

```

3273 \cs_new_protected:Npn \@@_insert_caption_i:
3274 {
3275   \group_begin:

```

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

```

3276     \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.

```

3277     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3278     \tl_if_empty:NTF \l_@@_short_caption_tl
3279     \caption
3280     { \caption [ \l_@@_short_caption_tl ] }
3281     { \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.

```

3282     \bool_if:NF \g_@@_caption_finished_bool
3283     {
3284       \bool_gset_true:N \g_@@_caption_finished_bool

```

```

3285     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3286     \int_gzero:N \c@tabularnote
3287   }
3288   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3289   \group_end:
3290 }
3291 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3292 {
3293   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3294   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3295 }
3296 \cs_set_protected:Npn \@@_tabular_notes_error:
3297 { \@@_error:n { Bad-use~of~NiceTabularNotes } }
3298 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3299 \cs_set_protected:Npn \@@_tabular_notes:
3300 {
3301   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3302   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3303   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3304   \skip_vertical:N 0.65ex

```

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

```

3305   \group_begin:
3306   \l_@@_notes_code_before_tl
3307   \tl_if_empty:NF \g_@@_tabularnote_tl
3308   {
3309     \g_@@_tabularnote_tl \par
3310     \tl_gclear:N \g_@@_tabularnote_tl
3311   }

```

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.

```

3312   \int_compare:nNnT \c@tabularnote > \c_zero_int
3313   {
3314     \bool_if:NTF \l_@@_notes_para_bool
3315     {
3316       \begin { tabularnotes* }
3317       \seq_map_inline:Nn \g_@@_notes_seq
3318       { \@@_one_tabularnote:nn ##1 }
3319       \strut
3320       \end { tabularnotes* }

```

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

```

3321     \par
3322   }
3323   {
3324     \tabularnotes
3325     \seq_map_inline:Nn \g_@@_notes_seq
3326     { \@@_one_tabularnote:nn ##1 }
3327     \strut
3328     \endtabularnotes
3329   }
3330 }
3331 \unskip
3332 \group_end:
3333 \bool_if:NT \l_@@_notes_bottomrule_bool
3334 {
3335   \IfPackageLoadedTF { booktabs }
3336   {

```

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

```

3337     \skip_vertical:N \aboverulesep

```

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

```

3338         { \CT@arc@ \hrule height \heavyrulewidth }
3339     }
3340     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3341 }
3342 \l_@@_notes_code_after_tl
3343 \seq_gclear:N \g_@@_notes_seq
3344 \seq_gclear:N \g_@@_notes_in_caption_seq
3345 \int_gzero:N \c@tabularnote
3346 }

```

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.

```

3347 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3348 {
3349     \tl_if_novalue:nTF { #1 }
3350     { \item }
3351     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3352 }

```

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.

```

3353 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3354 {
3355     \pgfpicture
3356     \@@_qpoint:n { row - 1 }
3357     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3358     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3359     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3360     \endpgfpicture
3361     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3362     \int_if_zero:nT \l_@@_first_row_int
3363     {
3364         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3365         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3366     }
3367     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3368 }

```

Now, the general case.

```

3369 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3370 {

```

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

```

3371     \str_if_eq:eeT \l_@@_baseline_tl { t }
3372     { \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`.

```

3373     \pgfpicture
3374     \@@_qpoint:n { row - 1 }
3375     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3376     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3377     {
3378         \int_set:Nn \l_tmpa_int
3379         {
3380             \str_range:Nnn
3381             \l_@@_baseline_tl
3382             { 6 }
3383             { \tl_count:o \l_@@_baseline_tl }

```

```

3384     }
3385     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3386   }
3387   {
3388     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3389     \bool_lazy_or:nnT
3390       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3391       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3392     {
3393       \@@_error:n { bad-value-for~baseline }
3394       \int_set:Nn \l_tmpa_int 1
3395     }
3396     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3397   }
3398   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3399   \endpgfpicture
3400   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3401   \int_if_zero:nT \l_@@_first_row_int
3402   {
3403     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3404     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3405   }
3406   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3407 }

```

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.

```

3408 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3409 {

```

We will compute the real width of both delimiters used.

```

3410   \dim_zero_new:N \l_@@_real_left_delim_dim
3411   \dim_zero_new:N \l_@@_real_right_delim_dim
3412   \hbox_set:Nn \l_tmpb_box
3413   {
3414     \m@th
3415     $ % $
3416     \left #1
3417     \vcenter
3418     {
3419       \vbox_to_ht:nn
3420         { \box_ht_plus_dp:N \l_tmpa_box }
3421       { }
3422     }
3423     \right .
3424     $ % $
3425   }
3426   \dim_set:Nn \l_@@_real_left_delim_dim
3427   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3428   \hbox_set:Nn \l_tmpb_box
3429   {
3430     \m@th
3431     $ % $
3432     \left .
3433     \vbox_to_ht:nn
3434       { \box_ht_plus_dp:N \l_tmpa_box }
3435     { }
3436     \right #2
3437     $ % $
3438   }
3439   \dim_set:Nn \l_@@_real_right_delim_dim
3440   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3441 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3442 \@@_put_box_in_flow:
3443 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3444 }

```

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).

```

3445 \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).

```

3446 {
3447   \peek_remove_spaces:n
3448   {
3449     \peek_meaning:NTF \end
3450     \@@_analyze_end:Nn
3451     {
3452       \@@_transform_preamble:
3453       \@@_array:o \g_@@_array_preamble_tl
3454     }
3455   }
3456 }
3457 {
3458   \@@_create_col_nodes:
3459   \endarray
3460 }

```

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

```

3461 \NewDocumentEnvironment { @@-light-syntax } { b }
3462 {

```

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`.

```

3463 \tl_if_empty:nT { #1 }
3464 { \@@_fatal:n { empty-environment } }
3465 \tl_if_in:nnT { #1 } { & }
3466 { \@@_fatal:n { ampersand-in-light-syntax } }
3467 \tl_if_in:nnT { #1 } { \ }
3468 { \@@_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`.

```

3469 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3470 }

```

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.

```

3471 {
3472   \@@_create_col_nodes:
3473   \endarray
3474 }

```

```

3475 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3476 {
3477   \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).

```

3478   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3479   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3480   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3481     \seq_set_split:Nee
3482     \seq_set_split:Non
3483     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3484   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3485   \tl_if_empty:NF \l_tmpa_tl
3486     { \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.

```

3487   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3488     { \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`).

```

3489   \tl_build_begin:N \l_@@_new_body_tl
3490   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3491   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3492   \@@_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).

```

3493   \seq_map_inline:Nn \l_@@_rows_seq
3494   {
3495     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3496     \@@_line_with_light_syntax:n { #1 }
3497   }
3498   \tl_build_end:N \l_@@_new_body_tl
3499   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3500   {
3501     \int_set:Nn \l_@@_last_col_int
3502     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3503   }

```

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.

```

3504   \@@_transform_preamble:

```

```

3505   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3506   }

```

```

3507 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3508 {
3509   \seq_clear_new:N \l_@@_cells_seq
3510   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3511   \int_set:Nn \l_@@_nb_cols_int
3512   { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3513   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3514   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3515   \seq_map_inline:Nn \l_@@_cells_seq

```

```

3516     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3517   }
3518   \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`.

```

3519   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3520   {
3521     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3522     { \@@_fatal:n { empty~environment } }

```

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

```

3523     \end { #2 }
3524   }

```

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`).

```

3525   \cs_new:Npn \@@_create_col_nodes:
3526   {
3527     \crrc
3528     \int_if_zero:nT \l_@@_first_col_int
3529     {
3530       \omit
3531       \hbox_overlap_left:n
3532       {
3533         \bool_if:NT \l_@@_code_before_bool
3534         { \pgfsys@markposition { \@@_env: - col - 0 } }
3535         \pgfpicture
3536         \pgfrememberpicturepositiononpagetrue
3537         \pgfcoordinate { \@@_env: - col - 0 } \pgfpointright
3538         \str_if_empty:NF \l_@@_name_str
3539         { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3540         \endpgfpicture
3541         \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3542       }
3543     }
3544   }
3545   \omit

```

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

```

3546   \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`).

```

3547   \int_if_zero:nTF \l_@@_first_col_int
3548   {
3549     \@@_mark_position:n { 1 }
3550     \pgfpicture
3551     \pgfrememberpicturepositiononpagetrue
3552     \pgfcoordinate { \@@_env: - col - 1 }
3553     { \pgfpointright { - 0.5 \arrayrulewidth } \c_zero_dim }
3554     \str_if_empty:NF \l_@@_name_str
3555     { \pgfnodelalias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3556     \endpgfpicture
3557   }
3558   {
3559     \bool_if:NT \l_@@_code_before_bool
3560     {
3561       \hbox
3562       {

```

```

3563         \skip_horizontal:n { 0.5 \arrayrulewidth }
3564         \pgfsys@markposition { \@@_env: - col - 1 }
3565         \skip_horizontal:n { -0.5 \arrayrulewidth }
3566     }
3567 }
3568 \pgfpicture
3569 \pgfrememberpicturepositiononpagetrue
3570 \pgfcoordinate { \@@_env: - col - 1 }
3571 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3572 \@@_node_alias:n { 1 }
3573 \endpgfpicture
3574 }

```

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.

```

3575 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3576 \bool_if:NF \l_@@_auto_columns_width_bool
3577 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3578 {
3579     \bool_lazy_and:nnTF
3580     \l_@@_auto_columns_width_bool
3581     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3582     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3583     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3584     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3585 }
3586 \skip_horizontal:N \g_tmpa_skip
3587 \hbox
3588 {
3589     \@@_mark_position:n { 2 }
3590     \pgfpicture
3591     \pgfrememberpicturepositiononpagetrue
3592     \pgfcoordinate { \@@_env: - col - 2 }
3593     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3594     \@@_node_alias:n { 2 }
3595     \endpgfpicture
3596 }

```

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.

```

3597 \int_gset:Nn \g_tmpa_int 1
3598 \bool_if:NTF \g_@@_last_col_found_bool
3599 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3600 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3601 {
3602     &
3603     \omit
3604     \int_gincr:N \g_tmpa_int

```

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

```

3605     \skip_horizontal:N \g_tmpa_skip
3606     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3607 \pgfpicture
3608 \pgfrememberpicturepositiononpagetrue
3609 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3610 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3611 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3612 \endpgfpicture
3613 }

```

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).

```

3614 \bool_lazy_or:nnF
3615 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3616 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3617 {
3618   &
3619   \omit
3620   \skip_horizontal:N \g_tmpa_skip
3621   \int_gincr:N \g_tmpa_int
3622   \bool_lazy_any:nF
3623   {
3624     \g_@@_delims_bool
3625     \l_@@_tabular_bool
3626     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3627     \l_@@_exterior_arraycolsep_bool
3628     \l_@@_bar_at_end_of_pream_bool
3629   }
3630   { \skip_horizontal:n { - \col@sep } }
3631   \bool_if:NT \l_@@_code_before_bool
3632   {
3633     \hbox
3634     {
3635       \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.

```

3636 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3637 { \skip_horizontal:n { - \arraycolsep } }
3638 \pgfsys@markposition
3639 { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3640 \skip_horizontal:n { 0.5 \arrayrulewidth }
3641 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3642 { \skip_horizontal:N \arraycolsep }
3643 }
3644 }
3645 \pgfpicture
3646 \pgfrememberpicturepositiononpagetrue
3647 \pgfcoordinate { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3648 {
3649   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3650   {
3651     \pgfpoint
3652     { - 0.5 \arrayrulewidth - \arraycolsep }
3653     \c_zero_dim
3654   }
3655   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3656 }
3657 @@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3658 \endpgfpicture
3659 }

3660 \bool_if:NT \g_@@_last_col_found_bool
3661 {
3662   \hbox_overlap_right:n
3663   {
3664     \skip_horizontal:N \g_@@_width_last_col_dim
3665     \skip_horizontal:N \col@sep
3666     \bool_if:NT \l_@@_code_before_bool
3667     {
3668       \pgfsys@markposition
3669       { @@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }

```

```

3670     }
3671     \pgfpicture
3672     \pgfrememberpicturepositiononpagetrue
3673     \pgfcoordinate
3674     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3675     \pgfpointorigin
3676     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3677     \endpgfpicture
3678   }
3679 }
3680 }

3681 \cs_new_protected:Npn \@@_mark_position:n #1
3682 {
3683   \bool_if:NT \l_@@_code_before_bool
3684   {
3685     \hbox
3686     {
3687       \skip_horizontal:n { -0.5 \arrayrulewidth }
3688       \pgfsys@markposition { \@@_env: - col - #1 }
3689       \skip_horizontal:n { 0.5 \arrayrulewidth }
3690     }
3691   }
3692 }

3693 \cs_new_protected:Npn \@@_node_alias:n #1
3694 {
3695   \str_if_empty:NF \l_@@_name_str
3696   { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3697 }

```

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

```

3698 \tl_const:Nn \c_@@_preamble_first_col_tl
3699 {
3700   >
3701   {

```

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

```

3702     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3703     \bool_gset_true:N \g_@@_after_col_zero_bool
3704     \@@_begin_of_row:
3705     \hbox_set:Nw \l_@@_cell_box
3706     \@@_math_toggle:
3707     \@@_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”.

```

3708     \int_compare:nNnT \c@iRow > \c_zero_int
3709     {
3710       \bool_lazy_or:nnT
3711       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3712       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3713       {
3714         \l_@@_code_for_first_col_tl
3715         \xglobal \colorlet { nicematrix-first-col } { . }
3716       }
3717     }
3718 }

```

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`.

```

3719     l
3720     <
3721     {
3722         \@@_math_toggle:
3723         \hbox_set_end:
3724         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3725         \@@_adjust_size_box:
3726         \@@_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.

```

3727         \dim_gset:Nn \g_@@_width_first_col_dim
3728         { \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.

```

3729         \hbox_overlap_left:n
3730         {
3731             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3732             \@@_node_cell:
3733             { \box_use_drop:N \l_@@_cell_box }
3734             \skip_horizontal:N \l_@@_left_delim_dim
3735             \skip_horizontal:N \l_@@_left_margin_dim
3736             \skip_horizontal:N \l_@@_extra_left_margin_dim
3737         }
3738         \bool_gset_false:N \g_@@_empty_cell_bool
3739         \skip_horizontal:n { -2 \col@sep }
3740     }
3741 }

```

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

```

3742 \tl_const:Nn \c_@@_preamble_last_col_tl
3743 {
3744     >
3745     {
3746         \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).

```

3747         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

3748         \bool_gset_true:N \g_@@_last_col_found_bool
3749         \int_gincr:N \c@jCol
3750         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3751         \hbox_set:Nw \l_@@_cell_box
3752         \@@_math_toggle:
3753         \@@_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”.

```

3754         \int_compare:nNnT \c@iRow > \c_zero_int
3755         {
3756             \bool_lazy_or:nnT
3757             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3758             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3759             {
3760                 \l_@@_code_for_last_col_tl
3761                 \xglobal \colorlet { nicematrix-last-col } { . }
3762             }
3763         }
3764     }
3765     l
3766     <

```

```

3767 {
3768   \@@_math_toggle:
3769   \hbox_set_end:
3770   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3771   \@@_adjust_size_box:
3772   \@@_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.

```

3773   \dim_gset:Nn \g_@@_width_last_col_dim
3774   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3775   \skip_horizontal:n { -2 \col@sep }

```

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

```

3776   \hbox_overlap_right:n
3777   {
3778     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3779     {
3780       \skip_horizontal:N \l_@@_right_delim_dim
3781       \skip_horizontal:N \l_@@_right_margin_dim
3782       \skip_horizontal:N \l_@@_extra_right_margin_dim
3783       \@@_node_cell:
3784     }
3785   }
3786   \bool_gset_false:N \g_@@_empty_cell_bool
3787 }
3788 }

```

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

```

3789 \NewDocumentEnvironment { NiceArray } { }
3790 {
3791   \bool_gset_false:N \g_@@_delims_bool
3792   \str_if_empty:NT \g_@@_name_env_str
3793   { \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).

```

3794   \NiceArrayWithDelims . .
3795 }
3796 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3797 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3798 {
3799   \NewDocumentEnvironment { #1 NiceArray } { }
3800   {
3801     \bool_gset_true:N \g_@@_delims_bool
3802     \str_if_empty:NT \g_@@_name_env_str
3803     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3804     \@@_test_if_math_mode:
3805     \NiceArrayWithDelims #2 #3
3806   }
3807   { \endNiceArrayWithDelims }
3808 }
3809 \@@_def_env:NNN p ( )
3810 \@@_def_env:NNN b [ ]
3811 \@@_def_env:NNN B \{ \}
3812 \@@_def_env:NNN v \vert \vert
3813 \@@_def_env:NNN V \Vert \Vert

```

## 13 The environment {NiceMatrix} and its variants

### 13.1 Definition of {pNiceMatrix}

```

3814 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3815 {
3816   \bool_set_false:N \l_@@_preamble_bool
3817   \tl_clear:N \l_tmpa_tl
3818   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3819     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3820   \tl_put_right:Nn \l_tmpa_tl
3821     {
3822       *
3823       {
3824         \int_case:nnF \l_@@_last_col_int
3825         {
3826           { -2 } { \c@MaxMatrixCols }
3827           { -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).

```

3828       }
3829       { \int_eval:n { \l_@@_last_col_int - 1 } }
3830     }
3831     { #2 }
3832   }
3833   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3834   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3835 }
3836 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3837 \clist_map_inline:nn { p , b , B , v , V }
3838 {
3839   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3840   {
3841     \bool_gset_true:N \g_@@_delims_bool
3842     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3843     \int_if_zero:nT \l_@@_last_col_int
3844     {
3845       \bool_set_true:N \l_@@_last_col_without_value_bool
3846       \int_set:Nn \l_@@_last_col_int { -1 }
3847     }
3848     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3849     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3850   }
3851   { \use:c { end #1 NiceArray } }
3852 }

```

We define also an environment {NiceMatrix}

```

3853 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3854 {
3855   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3856   \int_if_zero:nT \l_@@_last_col_int
3857   {
3858     \bool_set_true:N \l_@@_last_col_without_value_bool
3859     \int_set:Nn \l_@@_last_col_int { -1 }
3860   }
3861   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3862   \bool_lazy_or:nnT
3863     \l_@@_except_borders_bool
3864     { \clist_if_empty_p:N \l_@@_vlines_clist }
3865     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3866   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3867 }

```

```
3868 { \endNiceArray }
```

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

```
3869 \cs_new_protected:Npn \@@_NotEmpty:
3870 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

## 13.2 The key renew-matrix

```
3871 \cs_set_protected:Npn \@@_renew_matrix:
3872 {
3873   \tl_map_inline:nn { pvVbB }
3874   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3875 }
```

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

```
3876 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3877 {
```

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.

```
3878   \dim_compare:nNnT\l_@@_width_dim = \c_zero_dim
3879   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3880   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3881   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3882   \tl_if_empty:NF \l_@@_short_caption_tl
3883   {
3884     \tl_if_empty:NT \l_@@_caption_tl
3885     {
3886       \@@_error_or_warning:n { short-caption~without~caption }
3887       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3888     }
3889   }
3890   \tl_if_empty:NF \l_@@_label_tl
3891   {
3892     \tl_if_empty:NT \l_@@_caption_tl
3893     { \@@_error_or_warning:n { label~without~caption } }
3894   }
3895   \NewDocumentEnvironment { TabularNote } { b }
3896   {
3897     \bool_if:NTF \l_@@_in_code_after_bool
3898     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3899     {
3900       \tl_if_empty:NF \g_@@_tabularnote_tl
3901       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3902       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3903     }
3904   }
3905   { }
3906   \@@_settings_for_tabular:
3907   \NiceArray { #2 }
3908 }
3909 { \endNiceArray }
3910 \cs_new_protected:Npn \@@_settings_for_tabular:
3911 {
3912   \bool_set_true:N \l_@@_tabular_bool
3913   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3914   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3915   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3916 }
```

```

3917 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3918 {
3919   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3920   \dim_set:Nn \l_@@_width_dim { #1 }
3921   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3922   \@@_settings_for_tabular:
3923   \NiceArray { #3 }
3924 }
3925 {
3926   \endNiceArray
3927   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3928   { \@@_error:n { NiceTabularX~without~X } }
3929 }

3930 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3931 {
3932   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3933   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3934   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3935   \@@_settings_for_tabular:
3936   \NiceArray { #3 }
3937 }
3938 { \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`).

```

3939 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3940 {
3941   \bool_lazy_all:nT
3942   {
3943     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3944     { \l_@@_hvlines_bool }
3945     { ! \g_@@_delims_bool }
3946     { ! \l_@@_except_borders_bool }
3947   }
3948   {
3949     \bool_set_true:N \l_@@_except_borders_bool
3950     \clist_if_empty:NF \l_@@_corners_clist
3951     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3952     \tl_gput_right:Nn \g_@@_rules_tl
3953     {
3954       \@@_stroke_block:nnnnn
3955       {
3956         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3957         draw = \l_@@_rules_color_tl
3958       }
3959       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3960     }
3961   }
3962 }

3963 \cs_new_protected:Npn \@@_after_array:
3964 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` in order to come back to the standard definition of

`\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```
3965 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3966 \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.

```
3967 \bool_if:NT \g_@@_last_col_found_bool
3968 { \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`.

```
3969 \bool_if:NT \l_@@_last_col_without_value_bool
3970 { \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.

```
3971 \bool_if:NT \l_@@_last_row_without_value_bool
3972 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3973 \tl_gput_right:Ne \g_@@_aux_tl
3974 {
3975   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3976   {
3977     \int_use:N \l_@@_first_row_int ,
3978     \int_use:N \c@iRow ,
3979     \int_use:N \g_@@_row_total_int ,
3980     \int_use:N \l_@@_first_col_int ,
3981     \int_use:N \c@jCol ,
3982     \int_use:N \g_@@_col_total_int
3983   }
3984 }
3985 \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`).

```
3986 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3987 {
3988   \tl_gput_right:Ne \g_@@_aux_tl
3989   {
3990     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3991     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3992   }
3993 }
3994 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3995 {
3996   \tl_gput_right:Ne \g_@@_aux_tl
3997   {
3998     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3999     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4000     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4001     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4002   }
4003 }
```

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

```
4004 \@@_create_diag_nodes:
```

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

```

4005 \pgfpicture
4006 \@@_create_aliases_last:
4007 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4008 \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>12</sup>. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

4009 \bool_if:NT \l_@@_parallelize_diags_bool
4010 {
4011   \int_gzero:N \g_@@_ddots_int
4012   \int_gzero:N \g_@@_iddots_int

```

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

```

4013   \dim_gzero:N \g_@@_delta_x_one_dim
4014   \dim_gzero:N \g_@@_delta_y_one_dim
4015   \dim_gzero:N \g_@@_delta_x_two_dim
4016   \dim_gzero:N \g_@@_delta_y_two_dim
4017 }
4018 \bool_set_false:N \l_@@_initial_open_bool
4019 \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.

```

4020 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4021 \@@_draw_dotted_lines:

```

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

```

4022 \clist_if_empty:NF \l_@@_corners_clist
4023 {
4024   \bool_if:NTF \l_@@_no_cell_nodes_bool
4025   { \@@_error:n { corners~with~no~cell~nodes } }
4026   \@@_compute_corners:
4027 }

```

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

```

4028 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4029 \g_@@_pos_of_blocks_seq
4030 \g_@@_future_pos_of_blocks_seq
4031 \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-*}`).

```

4032 \@@_adjust_pos_of_blocks_seq:
4033 \@@_deal_with_rounded_corners:
4034 \legacy_if:nF { measuring@ } \@@_draw_rules:
4035 \tl_gclear:N \g_@@_rules_tl

```

---

<sup>12</sup>It’s possible to use the option `parallelize-diags` to disable this parallelization.

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

```

4036 \IfPackageLoadedT { tikz }
4037 {
4038   \tikzset
4039   {
4040     every-picture / .style =
4041     {
4042       overlay ,
4043       remember-picture ,
4044       name~prefix = \@@_env: -
4045     }
4046   }
4047 }
4048 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4049 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4050 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4051 \cs_set_eq:NN \OverBrace \@@_OverBrace
4052 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4053 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4054 \cs_set_eq:NN \line \@@_line
4055

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```

4056 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4057 \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.

```

4058 \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).

```

4059 \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`).

```

4060 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
4061 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }

```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```

4062 \bool_set_true:N \l_@@_in_code_after_bool
4063 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4064 \scan_stop:
4065 \tl_gclear:N \g_nicematrix_code_after_tl
4066 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4067 \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.

```

4068 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4069 \tl_if_empty:NF \g_@@_pre_code_before_tl
4070 {
4071   \tl_gput_right:Ne \g_@@_aux_tl
4072   {
4073     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4074     { \exp_not:o \g_@@_pre_code_before_tl }
4075   }

```

```

4076     \tl_gclear:N \g_@@_pre_code_before_tl
4077   }
4078   \tl_if_empty:NF \g_nicematrix_code_before_tl
4079   {
4080     \tl_gput_right:Ne \g_@@_aux_tl
4081     {
4082       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4083       { \exp_not:o \g_nicematrix_code_before_tl }
4084     }
4085     \tl_gclear:N \g_nicematrix_code_before_tl
4086   }

4087   \str_gclear:N \g_@@_name_env_str
4088   \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>13</sup>. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

4089   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4090 }

4091 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4092 {
4093   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4094   \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 corre-
spond to the options xdots/shorten-start and xdots/shorten-end available to the user.

4095   \dim_set:Nn \l_@@_xdots_shorten_start_dim
4096   { 0.6 \l_@@_xdots_shorten_start_dim }
4097   \dim_set:Nn \l_@@_xdots_shorten_end_dim
4098   { 0.6 \l_@@_xdots_shorten_end_dim }
4099 }

```

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`.

```

4100 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4101 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4102 \cs_new_protected:Npn \@@_create_alias_nodes:
4103 {
4104   \int_step_inline:nn \c@iRow
4105   {
4106     \pgfnodealias
4107     { \l_@@_name_str - ##1 - last }
4108     { \@@_env: - ##1 - \int_use:N \c@jCol }
4109   }
4110   \int_step_inline:nn \c@jCol
4111   {
4112     \pgfnodealias
4113     { \l_@@_name_str - last - ##1 }
4114     { \@@_env: - \int_use:N \c@iRow - ##1 }
4115   }
4116   \pgfnodealias
4117   { \l_@@_name_str - last - last }
4118   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4119 }

```

---

<sup>13</sup>e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

4120 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4121 {
4122   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4123   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4124 }

```

The following command must *not* be protected.

```

4125 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4126 {
4127   { #1 }
4128   { #2 }
4129   {
4130     \int_compare:nNnTF { #3 } > { 98 }
4131     { \int_use:N \c@iRow }
4132     { #3 }
4133   }
4134   {
4135     \int_compare:nNnTF { #4 } > { 98 }
4136     { \int_use:N \c@jCol }
4137     { #4 }
4138   }
4139   { #5 }
4140 }

```

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).

```

4141 \AtBeginDocument
4142 {
4143   \cs_new_protected:Npe \@@_draw_dotted_lines:
4144   {
4145     \c_@@_pgfortikzpicture_tl
4146     \@@_draw_dotted_lines_i:
4147     \c_@@_endpgfortikzpicture_tl
4148   }
4149 }

```

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

```

4150 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4151 {
4152   \pgfrememberpicturepositiononpagetrue
4153   \pgf@relevantforpicturesizefalse
4154   \g_@@_HVdotsfor_lines_tl
4155   \g_@@_Vdots_lines_tl
4156   \g_@@_Ddots_lines_tl
4157   \g_@@_Idots_lines_tl
4158   \g_@@_Cdots_lines_tl
4159   \g_@@_Ldots_lines_tl
4160 }

4161 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4162 {
4163   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4164   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4165 }

```

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

```

4166 \pgfdeclareshape { @@_diag_node }
4167 {
4168   \savedanchor { \five }
4169   {
4170     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4171     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4172   }
4173   \anchor { 5 } { \five }
4174   \anchor { center } { \pgfpointorigin }
4175   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4176   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4177   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4178   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4179   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4180   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4181   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4182   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4183   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4184   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4185 }

```

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).

```

4186 \cs_new_protected:Npn \@_create_diag_nodes:
4187 {
4188   \pgfpicture
4189   \pgfrememberpicturepositiononpagetrue
4190   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4191   {
4192     @@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4193     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4194     @@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4195     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4196     @@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4197     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4198     @@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4199     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4200     \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.

```

4201     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4202     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4203     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4204     \str_if_empty:NF \l_@@_name_str
4205     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4206   }

```

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

```

4207     \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4208     @@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4209     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4210     @@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4211     \pgfcoordinate
4212     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4213     \pgfnodealias
4214     { \@@_env: - last }
4215     { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4216     \str_if_empty:NF \l_@@_name_str
4217     {

```

```

4218     \pgfnodealias
4219     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4220     { \@@_env: - \int_use:N \l_tmpa_int }
4221     \pgfnodealias
4222     { \l_@@_name_str - last }
4223     { \@@_env: - last }
4224   }
4225 \endpgfpicture
4226 }

```

## 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.

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

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.

```

4227 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4228 {

```

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

```

4229 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4230 \l_@@_initial_i_int = #1
4231 \l_@@_initial_j_int = #2
4232 \l_@@_final_i_int = #1
4233 \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.

```

4234 \let \l_@@_stop_loop_bool \c_false_bool
4235 \bool_do_until:Nn \l_@@_stop_loop_bool
4236 {

```

We test if we are still in the matrix.

```

4237 \advance \l_@@_final_i_int by #3
4238 \advance \l_@@_final_j_int by #4
4239 \let \l_@@_final_open_bool \c_false_bool
4240 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4241 \if_int_compare:w #3 = \c_one_int
4242 \let \l_@@_final_open_bool \c_true_bool
4243 \else:
4244 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4245 \let \l_@@_final_open_bool \c_true_bool
4246 \fi:
4247 \fi:
4248 \else:
4249 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4250 \if_int_compare:w #4 = -1
4251 \let \l_@@_final_open_bool \c_true_bool
4252 \fi:
4253 \else:
4254 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4255 \if_int_compare:w #4 = \c_one_int
4256 \let \l_@@_final_open_bool \c_true_bool
4257 \fi:
4258 \fi:
4259 \fi:
4260 \fi:
4261 \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.

```

4262 {

```

We do a step backwards.

```

4263 \advance \l_@@_final_i_int by - #3
4264 \advance \l_@@_final_j_int by - #4
4265 \let \l_@@_stop_loop_bool \c_true_bool
4266 }

```

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`.

```

4267     {
4268         \cs_if_exist:cTF
4269         {
4270             @@ _ dotted _
4271             \int_use:N \l_@@_final_i_int -
4272             \int_use:N \l_@@_final_j_int
4273         }
4274         {
4275             \advance \l_@@_final_i_int by - #3
4276             \advance \l_@@_final_j_int by - #4
4277             \let \l_@@_final_open_bool \c_true_bool
4278             \let \l_@@_stop_loop_bool \c_true_bool
4279         }
4280     }
4281     \cs_if_exist:cTF
4282     {
4283         pgf @ sh @ ns @ \@@_env:
4284         - \int_use:N \l_@@_final_i_int
4285         - \int_use:N \l_@@_final_j_int
4286     }
4287     { \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.

```

4288     {
4289         \cs_set_nopar:cpn
4290         {
4291             @@ _ dotted _
4292             \int_use:N \l_@@_final_i_int -
4293             \int_use:N \l_@@_final_j_int
4294         }
4295         { }
4296     }
4297 }
4298 }
4299 }

```

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

```

4300     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4301     \l_tmpa_int = \l_@@_col_min_int
4302     \advance \l_tmpa_int by -1
4303     \bool_do_until:Nn \l_@@_stop_loop_bool
4304     {
4305         \advance \l_@@_initial_i_int by - #3
4306         \advance \l_@@_initial_j_int by - #4
4307         \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.

```

4308     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4309     \if_int_compare:w #3 = \c_one_int
4310     \let \l_@@_initial_open_bool \c_true_bool
4311     \else:

```

\l\_tmpa\_int contains \l\_@@\_col\_min\_int - 1 (only for efficiency).

```

4312         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4313         \let \l_@@_initial_open_bool \c_true_bool
4314     \fi:
4315 \fi:
4316 \else:
4317     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4318     \if_int_compare:w #4 = \c_one_int
4319     \let \l_@@_initial_open_bool \c_true_bool
4320     \fi:
4321 \else:
4322     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4323     \if_int_compare:w #4 = -1
4324     \let \l_@@_initial_open_bool \c_true_bool
4325     \fi:
4326 \fi:
4327 \fi:
4328 \fi:
4329 \bool_if:NTF \l_@@_initial_open_bool
4330 {
4331     \advance \l_@@_initial_i_int by #3
4332     \advance \l_@@_initial_j_int by #4
4333     \let \l_@@_stop_loop_bool \c_true_bool
4334 }
4335 {
4336     \cs_if_exist:cTF
4337     {
4338         @@ _ dotted _
4339         \int_use:N \l_@@_initial_i_int -
4340         \int_use:N \l_@@_initial_j_int
4341     }
4342     {
4343         \advance \l_@@_initial_i_int by #3
4344         \advance \l_@@_initial_j_int by #4
4345         \let \l_@@_initial_open_bool \c_true_bool
4346         \let \l_@@_stop_loop_bool \c_true_bool
4347     }
4348     {
4349         \cs_if_exist:cTF
4350         {
4351             pgf @ sh @ ns @ \@@_env:
4352             - \int_use:N \l_@@_initial_i_int
4353             - \int_use:N \l_@@_initial_j_int
4354         }
4355         { \let \l_@@_stop_loop_bool \c_true_bool }
4356         {
4357             \cs_set_nopar:cpn
4358             {
4359                 @@ _ dotted _
4360                 \int_use:N \l_@@_initial_i_int -
4361                 \int_use:N \l_@@_initial_j_int
4362             }
4363             { }
4364         }
4365     }
4366 }
4367 }
```

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.

```

4368 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4369 {
4370     { \int_use:N \l_@@_initial_i_int }
```

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

```

4371      { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4372      { \int_use:N \l_@@_final_i_int }
4373      { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4374      { }
4375    }
4376  }

```

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

```

4377 \cs_new_protected:Npn \@@_open_shorten:
4378 {
4379   \bool_if:NT \l_@@_initial_open_bool
4380     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4381   \bool_if:NT \l_@@_final_open_bool
4382     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4383 }

```

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).

```

4384 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4385 {
4386   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4387   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4388   \int_set_eq:NN \l_@@_row_max_int \c_iRow
4389   \int_set_eq:NN \l_@@_col_max_int \c_jCol

```

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`.

```

4390   \seq_if_empty:NF \g_@@_submatrix_seq
4391   {
4392     \seq_map_inline:Nn \g_@@_submatrix_seq
4393       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4394   }
4395 }

```

#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.

```

4396 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4397 {
4398   \if_int_compare:w #3 > #1
4399   \else:
4400     \if_int_compare:w #1 > #5
4401     \else:
4402       \if_int_compare:w #4 > #2
4403       \else:
4404         \if_int_compare:w #2 > #6
4405         \else:
4406           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4407           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4408           \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4409           \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4410         \fi:
4411       \fi:
4412     \fi:
4413   \fi:
4414 }

4415 \cs_new_protected:Npn \@@_set_initial_coords:
4416 {
4417   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4418   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4419 }
4420 \cs_new_protected:Npn \@@_set_final_coords:
4421 {
4422   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4423   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4424 }
4425 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4426 {
4427   \pgfpointanchor
4428   {
4429     \@@_env:
4430     - \int_use:N \l_@@_initial_i_int
4431     - \int_use:N \l_@@_initial_j_int
4432   }
4433   { #1 }
4434   \@@_set_initial_coords:
4435 }
4436 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4437 {
4438   \pgfpointanchor
4439   {
4440     \@@_env:
4441     - \int_use:N \l_@@_final_i_int
4442     - \int_use:N \l_@@_final_j_int
4443   }
4444   { #1 }
4445   \@@_set_final_coords:
4446 }

4447 \cs_new_protected:Npn \@@_open_x_initial_dim:
4448 {
4449   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4450   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4451   {
4452     \cs_if_exist:cT
4453     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4454     {
4455       \pgfpointanchor
4456       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }

```

```

4457         { west }
4458         \dim_set:Nn \l_@@_x_initial_dim
4459         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4460     }
4461 }

```

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

```

4462     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4463     {
4464         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4465         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4466         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4467     }
4468 }
4469 \cs_new_protected:Npn \@@_open_x_final_dim:
4470 {
4471     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4472     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4473     {
4474         \cs_if_exist:cT
4475         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4476         {
4477             \pgfpointanchor
4478             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4479             { east }
4480             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4481             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4482         }
4483     }

```

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

```

4484     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4485     {
4486         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4487         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4488         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4489     }
4490 }

```

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.

```

4491 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4492 {
4493     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4494     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4495     {
4496         \@@_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.

```

4497     \bool_if:NT \g_@@_aux_found_bool
4498     {
4499         {
4500             \@@_open_shorten:
4501             \int_if_zero:nTF { #1 }
4502             { \color { nicematrix-first-row } }
4503         }

```

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.

```

4504         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4505         { \color { nicematrix-last-row } }
4506     }

```

```

4507         \keys_set:nn { nicematrix / xdots } { #3 }
4508         \@@_color:o \l_@@_xdots_color_tl
4509         \@@_actually_draw_Ldots:
4510     }
4511 }
4512 }
4513 }

```

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

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

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

```

4514 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4515 {
4516     \bool_if:NTF \l_@@_initial_open_bool
4517     {
4518         \@@_open_x_initial_dim:
4519         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4520         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4521     }
4522     { \@@_set_initial_coords_from_anchor:n { base~east } }
4523     \bool_if:NTF \l_@@_final_open_bool
4524     {
4525         \@@_open_x_final_dim:
4526         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4527         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4528     }
4529     { \@@_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.

```

4530     \bool_lazy_all:nTF
4531     {
4532         \l_@@_initial_open_bool
4533         \l_@@_final_open_bool
4534         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4535     }
4536     {
4537         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4538         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4539     }

```

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

```

4540     {
4541         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4542         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4543     }
4544     \@@_draw_line:
4545 }

```

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.

```

4546 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4547 {
4548   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4549   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4550   {
4551     \@@_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.

```

4552     \bool_if:NT \g_@@_aux_found_bool
4553     {
4554       {
4555         \@@_open_shorten:
4556         \int_if_zero:nTF { #1 }
4557         { \color { nicematrix-first-row } }
4558         {

```

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.

```

4559         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4560         { \color { nicematrix-last-row } }
4561       }
4562       \keys_set:nn { nicematrix / xdots } { #3 }
4563       \@@_color:o \l_@@_xdots_color_tl
4564       \@@_actually_draw_Cdots:
4565     }
4566   }
4567 }
4568 }

```

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`.

```

4569 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4570 {
4571   \bool_if:NTF \l_@@_initial_open_bool
4572   \@@_open_x_initial_dim:
4573   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4574   \bool_if:NTF \l_@@_final_open_bool
4575   \@@_open_x_final_dim:
4576   { \@@_set_final_coords_from_anchor:n { mid-west } }
4577   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4578   {
4579     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4580     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4581     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4582     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4583     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4584   }
4585   {
4586     \bool_if:NT \l_@@_initial_open_bool
4587     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4588     \bool_if:NT \l_@@_final_open_bool

```

```

4589         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4590     }
4591     \@@_draw_line:
4592 }
4593 \cs_new_protected:Npn \@@_open_y_initial_dim:
4594 {
4595     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4596     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4597     {
4598         \cs_if_exist:cT
4599         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4600         {
4601             \pgfpointanchor
4602             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4603             { north }
4604             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4605             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4606         }
4607     }
4608     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4609     {
4610         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4611         \dim_set:Nn \l_@@_y_initial_dim
4612         {
4613             \fp_to_dim:n
4614             {
4615                 \pgf@y
4616                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4617             }
4618         }
4619     }
4620 }
4621 \cs_new_protected:Npn \@@_open_y_final_dim:
4622 {
4623     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4624     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4625     {
4626         \cs_if_exist:cT
4627         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4628         {
4629             \pgfpointanchor
4630             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4631             { south }
4632             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4633             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4634         }
4635     }
4636     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4637     {
4638         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4639         \dim_set:Nn \l_@@_y_final_dim
4640         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4641     }
4642 }

```

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.

```

4643 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4644 {
4645     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4646     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4647     {
4648         \@@_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.

```

4649     \bool_if:NT \g_@@_aux_found_bool
4650     {
4651     {
4652         \@@_open_shorten:
4653         \int_if_zero:nTF { #2 }
4654         { \color { nicematrix-first-col } }
4655         {
4656             \int_compare:nNt { #2 } = \l_@@_last_col_int
4657             { \color { nicematrix-last-col } }
4658         }
4659         \keys_set:nn { nicematrix / xdots } { #3 }
4660         \@@_color:o \l_@@_xdots_color_tl
4661         \bool_if:NTF \l_@@_Vbrace_bool
4662         \@@_actually_draw_Vbrace:
4663         \@@_actually_draw_Vdots:
4664     }
4665     }
4666 }
4667 }

```

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`.

```

4668 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4669 {
4670     \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4671     \@@_actually_draw_Vdots_i:
4672     \@@_actually_draw_Vdots_ii:
4673     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4674     \@@_draw_line:
4675 }

```

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

```

4676 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4677 {
4678     \@@_open_y_initial_dim:
4679     \@@_open_y_final_dim:
4680     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4681 {
4682     \@@_qpoint:n { col - 1 }
4683     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4684     \dim_sub:Nn \l_@@_x_initial_dim
4685     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4686 }
4687 {
4688     \bool_lazy_and:nnTF
4689     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4690     { \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”.

```

4691     {
4692         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4693         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4694         \dim_add:Nn \l_@@_x_initial_dim
4695             { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4696     }

```

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

```

4697     {
4698         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4699         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4700         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4701         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4702     }
4703 }
4704 }

```

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

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

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

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

```

4705 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4706 {
4707     \bool_set_false:N \l_tmpa_bool
4708     \bool_if:NF \l_@@_initial_open_bool
4709     {
4710         \bool_if:NF \l_@@_final_open_bool
4711         {
4712             \@@_set_initial_coords_from_anchor:n { south-west }
4713             \@@_set_final_coords_from_anchor:n { north-west }
4714             \bool_set:Nn \l_tmpa_bool
4715                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4716         }
4717     }

```

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

```

4718     \bool_if:NTF \l_@@_initial_open_bool
4719     {
4720         \@@_open_y_initial_dim:
4721         \@@_set_final_coords_from_anchor:n { north }
4722         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4723     }
4724     {
4725         \@@_set_initial_coords_from_anchor:n { south }
4726         \bool_if:NTF \l_@@_final_open_bool
4727         \@@_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.

```

4728     {
4729         \@@_set_final_coords_from_anchor:n { north }
4730         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4731         {
4732             \dim_set:Nn \l_@@_x_initial_dim
4733             {
4734                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4735                     \l_@@_x_initial_dim \l_@@_x_final_dim
4736             }
4737         }
4738     }
4739 }
4740 }

```

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`.

```

4741 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4742 {
4743   \bool_if:NTF \l_@@_initial_open_bool
4744     \@@_open_y_initial_dim:
4745     { \@@_set_initial_coords_from_anchor:n { south } }
4746   \bool_if:NTF \l_@@_final_open_bool
4747     \@@_open_y_final_dim:
4748     { \@@_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.

```

4749   \int_if_zero:nTF \l_@@_initial_j_int
4750   {
4751     \@@_qpoint:n { col - 1 }
4752     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4753     \dim_sub:Nn \l_@@_x_initial_dim
4754     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4755   }

```

Elsewhere, the brace must be drawn left flush.

```

4756   {
4757     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4758     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4759     \dim_add:Nn \l_@@_x_initial_dim
4760     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4761   }

```

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

```

4762   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4763   \@@_draw_line:
4764 }

```

```

4765 \cs_new:Npn \@@_colsep:
4766 { \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.

```

4767 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4768 {
4769   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4770   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4771   {
4772     \@@_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.

```

4773     \bool_if:NT \g_@@_aux_found_bool
4774     {
4775     {
4776         \@@_open_shorten:
4777         \keys_set:nn { nicematrix / xdots } { #3 }
4778         \@@_color:o \l_@@_xdots_color_tl
4779         \@@_actually_draw_Ddots:
4780     }
4781     }
4782 }
4783 }
```

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

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

```

4784 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4785 {
4786     \bool_if:NTF \l_@@_initial_open_bool
4787     {
4788         \@@_open_y_initial_dim:
4789         \@@_open_x_initial_dim:
4790     }
4791     { \@@_set_initial_coords_from_anchor:n { south~east } }
4792     \bool_if:NTF \l_@@_final_open_bool
4793     {
4794         \@@_open_x_final_dim:
4795         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4796     }
4797     { \@@_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.

```

4798     \bool_if:NT \l_@@_parallelize_diags_bool
4799     {
4800         \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).

```

4801         \int_compare:nNnTF \g_@@_ddots_int = 1
```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4802     {
4803         \dim_gset:Nn \g_@@_delta_x_one_dim
4804         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4805         \dim_gset:Nn \g_@@_delta_y_one_dim
4806         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4807     }
```

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`.

```

4808     {
4809         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4810         {
4811             \dim_set:Nn \l_@@_y_final_dim
4812             {
4813                 \l_@@_y_initial_dim +
4814                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4815                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4816             }
4817         }
4818     }
4819 }
4820 \@@_draw_line:
4821 }

```

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.

```

4822 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4823 {
4824     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4825     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4826     {
4827         \@@_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.

```

4828         \bool_if:NT \g_@@_aux_found_bool
4829         {
4830             {
4831                 \@@_open_shorten:
4832                 \keys_set:nn { nicematrix / xdots } { #3 }
4833                 \@@_color:o \l_@@_xdots_color_tl
4834                 \@@_actually_draw_Iddots:
4835             }
4836         }
4837     }
4838 }

```

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`.

```

4839 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4840 {
4841     \bool_if:NTF \l_@@_initial_open_bool
4842     {
4843         \@@_open_y_initial_dim:
4844         \@@_open_x_initial_dim:
4845     }
4846     { \@@_set_initial_coords_from_anchor:n { south-west } }
4847     \bool_if:NTF \l_@@_final_open_bool

```

```

4848 {
4849   \l_@@_open_y_final_dim:
4850   \l_@@_open_x_final_dim:
4851 }
4852 { \l_@@_set_final_coords_from_anchor:n { north-east } }
4853 \bool_if:NT \l_@@_parallelize_diags_bool
4854 {
4855   \int_gincr:N \g_@@_iddots_int
4856   \int_compare:nNnTF \g_@@_iddots_int = 1
4857   {
4858     \dim_gset:Nn \g_@@_delta_x_two_dim
4859     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4860     \dim_gset:Nn \g_@@_delta_y_two_dim
4861     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4862   }
4863   {
4864     \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4865     {
4866       \dim_set:Nn \l_@@_y_final_dim
4867       {
4868         \l_@@_y_initial_dim +
4869         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4870         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4871       }
4872     }
4873   }
4874 }
4875 \l_@@_draw_line:
4876 }

```

## 17 The actual instructions for drawing the dotted lines with TikZ

The command `\l_@@_draw_line:` has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4877 \cs_new_protected:Npn \l_@@_draw_line:
4878 {
4879   \pgfrememberpicturepositiononpagetrue
4880   \pgf@relevantforpicturesizefalse
4881   \bool_lazy_or:nnTF
4882     \l_@@_dotted_bool
4883     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4884     \l_@@_draw_standard_dotted_line:
4885     \l_@@_draw_unstandard_dotted_line:
4886 }

```

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.

```

4887 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4888 {
4889   \begin { scope }
4890     \@@_draw_unstandard_dotted_line:o
4891     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4892 }

```

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.

```

4893 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4894 {
4895   \@@_draw_unstandard_dotted_line:nooo
4896   { #1 }
4897   \l_@@_xdots_up_tl
4898   \l_@@_xdots_down_tl
4899   \l_@@_xdots_middle_tl
4900 }
4901 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4902 \AtBeginDocument
4903 {
4904   \IfPackageLoadedT { tikz }
4905   {
4906     \tikzset
4907     {
4908       @@_node_above / .style = { sloped , above } ,
4909       @@_node_below / .style = { sloped , below } ,
4910       @@_node_middle / .style =
4911       {
4912         sloped ,
4913         inner~sep = \c_@@_innersep_middle_dim
4914       }
4915     }
4916   }
4917 }

4918 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4919 {

```

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

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

```

4920 \dim_zero_new:N \l_@@_l_dim
4921 \dim_set:Nn \l_@@_l_dim
4922 {
4923   \fp_to_dim:n
4924   {
4925     sqrt
4926     (
4927       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4928       +
4929       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4930     )
4931   }
4932 }

```

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.

```

4933   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4934   {
4935     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4936     \@@_draw_unstandard_dotted_line_i:
4937   }

```

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

```

4938   \bool_if:NT \l_@@_xdots_h_labels_bool
4939   {
4940     \tikzset
4941     {
4942       @@_node_above / .style = { auto = left } ,
4943       @@_node_below / .style = { auto = right } ,
4944       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4945     }
4946   }
4947   \tl_if_empty:nF { #4 }
4948   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4949   \dim_zero:N \l_tmpa_dim
4950   \dim_zero:N \l_tmpb_dim
4951   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4952   {

```

We test whether the brace is vertical or horizontal.

```

4953     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4954     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4955     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4956   }
4957   {
4958     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4959     {
4960       \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4961       { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4962       { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4963     }
4964   }
4965   \use:e
4966   {
4967     \exp_not:N \begin { scope }
4968     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
4969   }
4970   \draw
4971   [ #1 ]
4972   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4973   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4974   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4975   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4976   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4977   \end { scope }
4978   \end { scope }
4979 }
4980 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4981 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4982 {
4983   \dim_set:Nn \l_tmpa_dim
4984   {
4985     \l_@@_x_initial_dim
4986     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4987     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4988     }
4989     \dim_set:Nn \l_tmpb_dim
4990     {
4991         \l_@@_y_initial_dim
4992         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4993         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4994     }
4995     \dim_set:Nn \l_@@_tmpc_dim
4996     {
4997         \l_@@_x_final_dim
4998         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4999         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5000     }
5001     \dim_set:Nn \l_@@_tmpd_dim
5002     {
5003         \l_@@_y_final_dim
5004         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5005         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5006     }
5007     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5008     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5009     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5010     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5011 }

```

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

```

5012 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5013 {
5014 {

```

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

```

5015     \dim_zero_new:N \l_@@_l_dim
5016     \dim_set:Nn \l_@@_l_dim
5017     {
5018         \fp_to_dim:n
5019         {
5020             sqrt
5021             (
5022                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5023                 +
5024                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5025             )
5026         }
5027     }

```

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.

```

5028     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5029     {
5030         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5031         \@@_draw_standard_dotted_line_i:
5032     }
5033 }
5034 \bool_lazy_all:nF
5035 {
5036     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5037     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5038     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5039 }

```

```

5040 \@@_labels_standard_dotted_line:
5041 }
5042 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5043 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5044 {

```

The number of dots will be  $\l_1\text{tmpa\_int} + 1$ .

```

5045 \int_set:Nn \l_1tmpa_int
5046 {
5047 \dim_ratio:nn
5048 {
5049 \l_@@_l_dim
5050 - \l_@@_xdots_shorten_start_dim
5051 - \l_@@_xdots_shorten_end_dim
5052 }
5053 \l_@@_xdots_inter_dim
5054 }

```

The dimensions  $\l_1\text{tmpa\_dim}$  and  $\l_1\text{tmpb\_dim}$  are the coordinates of the vector between two dots in the dotted line.

```

5055 \dim_set:Nn \l_1tmpa_dim
5056 {
5057 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5058 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5059 }
5060 \dim_set:Nn \l_1tmpb_dim
5061 {
5062 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5063 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5064 }

```

In the loop over the dots, the dimensions  $\l_1\text{@@_x\_initial\_dim}$  and  $\l_1\text{@@_y\_initial\_dim}$  will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5065 \dim_gadd:Nn \l_@@_x_initial_dim
5066 {
5067 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5068 \dim_ratio:nn
5069 {
5070 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5071 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5072 }
5073 { 2 \l_@@_l_dim }
5074 }
5075 \dim_gadd:Nn \l_@@_y_initial_dim
5076 {
5077 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5078 \dim_ratio:nn
5079 {
5080 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
5081 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5082 }
5083 { 2 \l_@@_l_dim }
5084 }
5085 \pgf@relevantforpicturesizefalse
5086 \int_step_inline:nnn \c_zero_int \l_1tmpa_int
5087 {
5088 \pgfpathcircle
5089 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5090 \l_@@_xdots_radius_dim
5091 \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
5092 \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
5093 }
5094 \pgfusepathqfill
5095 }

```

```

5096 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5097 {
5098   \pgfscope
5099   \pgftransformshift
5100   {
5101     \pgfpointlineattime { 0.5 }
5102     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5103     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5104   }
5105   \fp_set:Nn \l_tmpa_fp
5106   {
5107     atand
5108     (
5109       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5110       \l_@@_x_final_dim - \l_@@_x_initial_dim
5111     )
5112   }
5113   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5114   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5115   \tl_if_empty:NF \l_@@_xdots_middle_tl
5116   {
5117     \begin { pgfscope }
5118     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5119     \pgfnode
5120     { rectangle }
5121     { center }
5122     {
5123       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5124       {
5125         $ % $
5126         \scriptstyle \l_@@_xdots_middle_tl
5127         $ % $
5128       }
5129     }
5130     { }
5131     {
5132       \pgfsetfillcolor { white }
5133       \pgfusepath { fill }
5134     }
5135     \end { pgfscope }
5136   }
5137   \tl_if_empty:NF \l_@@_xdots_up_tl
5138   {
5139     \pgfnode
5140     { rectangle }
5141     { south }
5142     {
5143       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5144       {
5145         $ % $
5146         \scriptstyle \l_@@_xdots_up_tl
5147         $ % $
5148       }
5149     }
5150     { }
5151     { \pgfusepath { } }
5152   }
5153   \tl_if_empty:NF \l_@@_xdots_down_tl
5154   {
5155     \pgfnode
5156     { rectangle }
5157     { north }
5158     {

```

```

5159         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5160         {
5161             $ % $
5162             \scriptstyle \l_@@_xdots_down_tl
5163             $ % $
5164         }
5165     }
5166     { }
5167     { \pgfusepath { } }
5168 }
5169 \endpgfscope
5170 }

```

## 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.

```

5171 \AtBeginDocument
5172 {

```

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

```

5173     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5174     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5175     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5176     {
5177         \int_if_zero:nTF \c@jCol
5178         { \@@_error:nn { in~first~col } { \Ldots } }
5179         {
5180             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5181             { \@@_error:nn { in~last~col } { \Ldots } }
5182             {
5183                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5184                 { #1 , down = #2 , up = #3 , middle = #4 }
5185             }
5186         }
5187         \bool_if:NF \l_@@_nullify_dots_bool
5188         { \phantom { \ensuremath { \@@_old_ldots: } } }
5189         \bool_gset_true:N \g_@@_empty_cell_bool
5190     }

5191     \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5192     \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5193     {
5194         \int_if_zero:nTF \c@jCol
5195         { \@@_error:nn { in~first~col } { \Cdots } }
5196         {
5197             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5198             { \@@_error:nn { in~last~col } { \Cdots } }
5199             {
5200                 \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }

```

```

5201         { #1 , down = #2 , up = #3 , middle = #4 }
5202     }
5203 }
5204 \bool_if:NF \l_@@_nullify_dots_bool
5205 { \phantom { \ensuremath { \@@_old_cdots: } } }
5206 \bool_gset_true:N \g_@@_empty_cell_bool
5207 }

5208 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5209 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5210 {
5211     \int_if_zero:nTF \c@iRow
5212     { \@@_error:nn { in~first~row } { \Vdots } }
5213     {
5214         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5215         { \@@_error:nn { in~last~row } { \Vdots } }
5216         {
5217             \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5218             { #1 , down = #2 , up = #3 , middle = #4 }
5219         }
5220     }
5221     \bool_if:NF \l_@@_nullify_dots_bool
5222     { \phantom { \ensuremath { \@@_old_vdots: } } }
5223     \bool_gset_true:N \g_@@_empty_cell_bool
5224 }

5225 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5226 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5227 {
5228     \int_case:nnF \c@iRow
5229     {
5230         0 { \@@_error:nn { in~first~row } { \Ddots } }
5231         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5232     }
5233     {
5234         \int_case:nnF \c@jCol
5235         {
5236             0 { \@@_error:nn { in~first~col } { \Ddots } }
5237             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5238         }
5239         {
5240             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5241             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5242             { #1 , down = #2 , up = #3 , middle = #4 }
5243         }
5244     }
5245 }
5246 \bool_if:NF \l_@@_nullify_dots_bool
5247 { \phantom { \ensuremath { \@@_old_ddots: } } }
5248 \bool_gset_true:N \g_@@_empty_cell_bool
5249 }

5250 \cs_new_protected:Npn \@@_Iddots:
5251 { \@@_collect_options:n { \@@_Iddots_i } }
5252 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5253 {
5254     \int_case:nnF \c@iRow
5255     {
5256         0 { \@@_error:nn { in~first~row } { \Iddots } }
5257         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5258     }

```

```

5259     {
5260         \int_case:nnF \c@jCol
5261         {
5262             0 { \@@_error:nn { in~first~col } { \Iddots } }
5263             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5264         }
5265         {
5266             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5267             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5268             { #1 , down = #2 , up = #3 , middle = #4 }
5269         }
5270     }
5271     \bool_if:NF \l_@@_nullify_dots_bool
5272     { \phantom { \ensuremath { \@@_old_iddots: } } }
5273     \bool_gset_true:N \g_@@_empty_cell_bool
5274 }
5275 }

```

End of the \AddToHook.

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

```

5276 \keys_define:nn { nicematrix / Ddots }
5277 {
5278     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5279     draw-first .value_forbidden:n = true ,
5280 }

```

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

```

5281 \cs_new_protected:Npn \@@_Hspace:
5282 {
5283     \bool_gset_true:N \g_@@_empty_cell_bool
5284     \hspace
5285 }

```

In the environments of `nicematrix`, the command `\multicolumn` is redefined. We will patch the environment `{tabular}` to go back to the previous value of `\multicolumn`.

```

5286 \cs_new_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. TikZ nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5287 \cs_new:Npn \@@_Hdotsfor:
5288 {
5289     \bool_lazy_and:nnTF
5290     { \int_if_zero_p:n \c@jCol }
5291     { \int_if_zero_p:n \l_@@_first_col_int }
5292     {
5293         \bool_if:NTF \g_@@_after_col_zero_bool
5294         {
5295             \multicolumn { 1 } { c } { }
5296             \@@_Hdotsfor_i:
5297         }
5298         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5299     }
5300 }
5301 \multicolumn { 1 } { c } { }
5302 \@@_Hdotsfor_i:
5303 }
5304 }

```

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:`).

```
5305 \AtBeginDocument
5306 {
```

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

```
5307 \cs_new_protected:Npn \@@_Hdotsfor_i:
5308 { \@@_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`).

```
5309 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5310 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5311 {
5312   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5313   {
5314     \@@_Hdotsfor:nnnn
5315     { \int_use:N \c@iRow }
5316     { \int_use:N \c@jCol }
5317     { #2 }
5318     {
5319       #1 , #3 ,
5320       down = \exp_not:n { #4 } ,
5321       up = \exp_not:n { #5 } ,
5322       middle = \exp_not:n { #6 }
5323     }
5324   }
5325   \prg_replicate:nn { #2 - 1 }
5326   {
5327     &
5328     \multicolumn { 1 } { c } { }
5329     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5330   }
5331 }
5332 }
```

```
5333 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5334 {
5335   \bool_set_false:N \l_@@_initial_open_bool
5336   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5337 \int_set:Nn \l_@@_initial_i_int { #1 }
5338 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5339 \int_compare:nNnTF { #2 } = 1
5340 {
5341   \int_set:Nn \l_@@_initial_j_int 1
5342   \bool_set_true:N \l_@@_initial_open_bool
5343 }
5344 {
5345   \cs_if_exist:cTF
5346   {
5347     pgf @ sh @ ns @ \@@_env:
5348     - \int_use:N \l_@@_initial_i_int
5349     - \int_eval:n { #2 - 1 }
5350   }
5351   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5352   {
5353     \int_set:Nn \l_@@_initial_j_int { #2 }
5354     \bool_set_true:N \l_@@_initial_open_bool
```

```

5355     }
5356   }
5357   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5358   {
5359     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5360     \bool_set_true:N \l_@@_final_open_bool
5361   }
5362   {
5363     \cs_if_exist:cTF
5364     {
5365       pgf @ sh @ ns @ \@@_env:
5366       - \int_use:N \l_@@_final_i_int
5367       - \int_eval:n { #2 + #3 }
5368     }
5369     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5370     {
5371       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5372       \bool_set_true:N \l_@@_final_open_bool
5373     }
5374   }
5375   \bool_if:NT \g_@@_aux_found_bool
5376   {
5377     {
5378       \@@_open_shorten:
5379       \int_if_zero:nTF { #1 }
5380       { \color { nicematrix-first-row } }
5381       {
5382         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5383         { \color { nicematrix-last-row } }
5384       }
5385       \keys_set:nn { nicematrix / xdots } { #4 }
5386       \@@_color:o \l_@@_xdots_color_tl
5387       \@@_actually_draw_Ldots:
5388     }
5389   }

```

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).

```

5390   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5391   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5392 }

```

```

5393 \AtBeginDocument
5394 {
5395   \cs_new_protected:Npn \@@_Vdotsfor:
5396   { \@@_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`).

```

5397   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5398   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5399   {
5400     \bool_gset_true:N \g_@@_empty_cell_bool
5401     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5402     {
5403       \@@_Vdotsfor:nnnn
5404       { \int_use:N \c@iRow }
5405       { \int_use:N \c@jCol }
5406       { #2 }
5407       {
5408         #1 , #3 ,

```

```

5409         down = \exp_not:n { #4 } ,
5410         up = \exp_not:n { #5 } ,
5411         middle = \exp_not:n { #6 }
5412     }
5413 }
5414 }
5415 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5416 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5417 {
5418     \bool_set_false:N \l_@@_initial_open_bool
5419     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5420     \int_set:Nn \l_@@_initial_j_int { #2 }
5421     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5422     \int_compare:nNnTF { #1 } = 1
5423     {
5424         \int_set:Nn \l_@@_initial_i_int 1
5425         \bool_set_true:N \l_@@_initial_open_bool
5426     }
5427     {
5428         \cs_if_exist:cTF
5429         {
5430             pgf @ sh @ ns @ \@@_env:
5431             - \int_eval:n { #1 - 1 }
5432             - \int_use:N \l_@@_initial_j_int
5433         }
5434         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5435         {
5436             \int_set:Nn \l_@@_initial_i_int { #1 }
5437             \bool_set_true:N \l_@@_initial_open_bool
5438         }
5439     }
5440     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5441     {
5442         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5443         \bool_set_true:N \l_@@_final_open_bool
5444     }
5445     {
5446         \cs_if_exist:cTF
5447         {
5448             pgf @ sh @ ns @ \@@_env:
5449             - \int_eval:n { #1 + #3 }
5450             - \int_use:N \l_@@_final_j_int
5451         }
5452         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5453         {
5454             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5455             \bool_set_true:N \l_@@_final_open_bool
5456         }
5457     }
5458     \bool_if:NT \g_@@_aux_found_bool
5459     {
5460         {
5461             \@@_open_shorten:
5462             \int_if_zero:nTF { #2 }
5463             { \color { nicematrix-first-col } }

```

```

5464         {
5465             \int_compare:nNtT { #2 } = \g_@@_col_total_int
5466             { \color { nicematrix-last-col } }
5467         }
5468         \keys_set:nn { nicematrix / xdots } { #4 }
5469         \@@_color:o \l_@@_xdots_color_tl
5470         \bool_if:NTF \l_@@_Vbrace_bool
5471             \@@_actually_draw_Vbrace:
5472             \@@_actually_draw_Vdots:
5473     }
5474 }

```

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).

```

5475     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5476     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5477 }

```

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

```

5478 \NewDocumentCommand \@@_rotate: { 0 { } }
5479 {
5480     \bool_gset_true:N \g_@@_rotate_bool
5481     \keys_set:nn { nicematrix / rotate } { #1 }
5482     \ignorespaces
5483 }

```

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

```

5484 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate-in-p-col } }

5485 \keys_define:nn { nicematrix / rotate }
5486 {
5487     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5488     c .value_forbidden:n = true ,
5489     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5490 }

```

## 19 The command `\line` accessible in `\CodeAfter`

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format *i-j*) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format *i-j*, our command applies the command `\int_eval:n` to *i* and *j* ;
- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).<sup>14</sup>

```

5491 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5492 {
5493   \tl_if_empty:nTF { #2 }
5494     { #1 }
5495     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5496 }
5497 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5498 { \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).

```

5499 \AtBeginDocument
5500 {

```

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

```

5501   \tl_set_rescan:Nnn \l_tmpa_tl { }
5502   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5503   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5504   {
5505     {
5506       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5507       \@@_color:o \l_@@_xdots_color_tl
5508       \use:e
5509       {
5510         \@@_line_i:nn
5511         { \@@_double_int_eval:n #2 - \q_stop }
5512         { \@@_double_int_eval:n #3 - \q_stop }
5513       }
5514     }
5515   }
5516 }

5517 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5518 {
5519   \bool_set_false:N \l_@@_initial_open_bool
5520   \bool_set_false:N \l_@@_final_open_bool
5521   \bool_lazy_or:nnTF
5522     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5523     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5524     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5525   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5526 }

5527 \AtBeginDocument
5528 {
5529   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5530   {

```

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:`.

```

5531   \c_@@_pgfortikzpicture_tl
5532   \@@_draw_line_iii:nn { #1 } { #2 }
5533   \c_@@_endpgfortikzpicture_tl
5534 }
5535 }

```

---

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

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

```

5536 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5537 {
5538   \pgfrememberpicturepositiononpagetrue
5539   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5540   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5541   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5542   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5543   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5544   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5545   \@@_draw_line:
5546 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```

5547 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5548 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }

```

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

```

5549 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5550 {
5551   \tl_gput_right:Ne \g_@@_row_style_tl
5552   {

```

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

```

5553   \exp_not:N
5554   \@@_if_row_less_than:nn
5555   { \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`).

```

5556   {
5557     \exp_not:N
5558     \@@_if_col_greater_than:nn
5559     { \int_use:N \c@jCol }
5560     { \exp_not:n { #1 } \scan_stop: }
5561   }
5562 }
5563 }
5564 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5565 \keys_define:nn { nicematrix / RowStyle }
5566 {
5567   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5568   cell-space-top-limit+ .code:n =
5569     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5570   cell-space-top-limit+ .value_required:n = true ,
5571   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5572   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5573   cell-space-bottom-limit+ .code:n =
5574     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5575   cell-space-bottom-limit+ .value_required:n = true ,
5576   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5577   cell-space-limits .meta:n =
5578     {
5579       cell-space-top-limit = #1 ,
5580       cell-space-bottom-limit = #1 ,
5581     } ,
5582   cell-space-limits+ .meta:n =
5583     {
5584       cell-space-top-limit += #1 ,
5585       cell-space-bottom-limit += #1 ,
5586     } ,
5587   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5588   color .tl_set:N = \l_@@_color_tl ,
5589   color .value_required:n = true ,
5590   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5591   nb-rows .code:n =
5592     \str_if_eq:eeTF { #1 } { * }
5593     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5594     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5595   nb-rows .value_required:n = true ,
5596   fill .tl_set:N = \l_@@_fill_tl ,
5597   fill .value_required:n = true ,

```

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

```

5598   opacity .tl_set:N = \l_@@_opacity_tl ,
5599   opacity .value_required:n = true ,
5600   rowcolor .tl_set:N = \l_@@_fill_tl ,
5601   rowcolor .value_required:n = true ,
5602   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5603   rounded-corners .default:n = 4 pt ,
5604   unknown .code:n =
5605     \@@_unknown_key:nn
5606     { nicematrix / RowStyle }
5607     { Unknown-key-for-RowStyle }
5608 }

```

```

5609 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5610 {
5611   \group_begin:
5612   \tl_clear:N \l_@@_fill_tl
5613   \tl_clear:N \l_@@_opacity_tl
5614   \tl_clear:N \l_@@_color_tl
5615   \int_set:Nn \l_@@_key_nb_rows_int 1
5616   \dim_zero:N \l_@@_rounded_corners_dim
5617   \dim_zero:N \l_tmpa_dim
5618   \dim_zero:N \l_tmpb_dim
5619   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5620   \tl_if_empty:NF \l_@@_fill_tl
5621   {
5622     \@@_add_opacity_to_fill:

```

```

5623     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5624     {
The command \@@_exp_color_arg:No is fully expandable.
5625     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5626     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5627     {
5628     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5629     - *
5630     }
5631     { \dim_use:N \l_@@_rounded_corners_dim }
5632     }
5633     }
5634     \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5635     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5636     {
5637     \@@_put_in_row_style:e
5638     {
5639     \@@_put_in_cell_after_hook:n
5640     {

```

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

```

5641     \dim_set:Nn \l_@@_cell_space_top_limit_dim
5642     { \dim_use:N \l_tmpa_dim }
5643     }
5644     }
5645     }

```

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

```

5646     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5647     {
5648     \@@_put_in_row_style:e
5649     {
5650     \@@_put_in_cell_after_hook:n
5651     {
5652     \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5653     { \dim_use:N \l_tmpb_dim }
5654     }
5655     }
5656     }

```

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

```

5657     \tl_if_empty:NF \l_@@_color_tl
5658     {
5659     \@@_put_in_row_style:e
5660     {
5661     \mode_leave_vertical:
5662     \@@_color:n { \l_@@_color_tl }
5663     }
5664     }

```

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

```

5665     \bool_if:NT \l_@@_bold_row_style_bool
5666     {
5667     \@@_put_in_row_style:n
5668     {
5669     \exp_not:n
5670     {
5671     \if_mode_math:
5672     $ % $
5673     \bfseries \boldmath
5674     $ % $
5675     \else:
5676     \bfseries \boldmath

```

```

5677         \fi:
5678     }
5679 }
5680 }
5681 \group_end:
5682 \g_@@_row_style_tl
5683 \ignorespaces
5684 }

```

The following commande must *not* be protected.

```

5685 \cs_new:Npn \@@_rounded_from_row:n #1
5686 {
5687     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5688     { \int_eval:n { #1 } - 1 }
5689     {
5690         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5691         - \exp_not:n { \int_use:N \c@jCol }
5692     }
5693     { \dim_use:N \l_@@_rounded_corners_dim }
5694 }

```

## 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).

```

5695 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5696 {

```

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.

```

5697     \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`.

```

5698     \str_if_in:nnF { #1 } { !! }
5699     {
5700         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```
5701     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5702   }
5703   \int_if_zero:nTF \l_tmpa_int
```

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

```
5704   {
5705     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5706     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5707   }
```

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

```
5708     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5709   }
5710   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
```

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

```
5711 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5712 {
5713   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5714   {
```

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

```
5715     \group_begin:
5716     \pgfsetcornersarced
5717     { \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.

```
5718     \bool_if:NTF \l_@@_hvlines_bool
5719     {
5720       \pgfpathrectanglecorners
5721       {
5722         \pgfpointadd
5723         { \@@_qpoint:n { row-1 } }
5724         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5725       }
5726       {
5727         \pgfpointadd
5728         {
5729           \@@_qpoint:n
5730           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5731         }
5732         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5733       }
5734     }
5735     {
5736       \pgfpathrectanglecorners
5737       { \@@_qpoint:n { row-1 } }
5738       {
5739         \pgfpointadd
5740         {
5741           \@@_qpoint:n
5742           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5743         }
5744         { \pgfpoint \c_zero_dim \arrayrulewidth }
5745       }
5746     }
5747     \pgfusepath { clip }
5748     \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```
5749     }
5750 }
```

The macro `\@@_actually_color:` will 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`).

```
5751 \cs_new_protected:Npn \@@_actually_color:
5752 {
5753   \pgfpicture
5754   \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.

```
5755   \@@_clip_with_rounded_corners:
5756   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5757   {
5758     \int_compare:nNnTF { ##1 } = 1
5759     {
5760       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5761       \use:c { g_@@_color _ 1 _tl }
5762       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5763     }
5764     {
5765       \begin { pgfscope }
5766         \@@_color_opacity: ##2
5767         \use:c { g_@@_color _ ##1 _tl }
5768         \tl_gclear:c { g_@@_color _ ##1 _tl }
5769         \pgfusepath { fill }
5770       \end { pgfscope }
5771     }
5772   }
5773   \endpgfpicture
5774 }
```

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

```
5775 \cs_new_protected:Npn \@@_color_opacity:
5776 {
5777   \peek_meaning:NTF [
5778     \@@_color_opacity:w
5779     { \@@_color_opacity:w [ ] }
5780   }
```

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.

```
5781 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5782 {
5783   \tl_clear:N \l_tmpa_tl
5784   \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.

```
5785   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5786   \tl_if_empty:NTF \l_tmpb_tl
5787     \@declaredcolor
5788     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5789 }
```

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

```
5790 \keys_define:nn { nicematrix / color-opacity }
5791 {
```

```

5792     opacity .tl_set:N      = \l_tmpa_tl ,
5793     opacity .value_required:n = true
5794 }

```

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

```

5795 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5796 {
5797     \def \l_@@_rows_tl { #1 }
5798     \def \l_@@_cols_tl { #2 }
5799     \@@_cartesian_path:
5800 }

```

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

```

5801 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5802 {
5803     \tl_if_blank:nF { #2 }
5804     {
5805         \@@_add_to_colors_seq:en
5806         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5807         { \@@_cartesian_color:nn { #3 } { - } }
5808     }
5809 }

```

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

```

5810 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5811 {
5812     \tl_if_blank:nF { #2 }
5813     {
5814         \@@_add_to_colors_seq:en
5815         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5816         { \@@_cartesian_color:nn { - } { #3 } }
5817     }
5818 }

```

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

```

5819 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5820 {
5821     \tl_if_blank:nF { #2 }
5822     {
5823         \@@_add_to_colors_seq:en
5824         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5825         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5826     }
5827 }

```

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

```

5828 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5829 {
5830     \tl_if_blank:nF { #2 }
5831     {
5832         \@@_add_to_colors_seq:en
5833         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5834         { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5835     }
5836 }

```

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

```

5837 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5838 {
5839     \@@_cut_on_hyphen:w #1 \q_stop

```

```

5840 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5841 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5842 \@@_cut_on_hyphen:w #2 \q_stop
5843 \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5844 \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`.

```

5845 \@@_cartesian_path:n { #3 }
5846 }

```

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

```

5847 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5848 {
5849   \clist_map_inline:nn { #3 }
5850     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5851 }

```

```

5852 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5853 {
5854   \int_step_inline:nn \c@iRow
5855   {
5856     \int_step_inline:nn \c@jCol
5857     {
5858       \int_if_even:nTF { #####1 + ##1 }
5859         { \@@_cellcolor [ #1 ] { #2 } }
5860         { \@@_cellcolor [ #1 ] { #3 } }
5861       { ##1 - #####1 }
5862     }
5863   }
5864 }

```

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”.

```

5865 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5866 {
5867   \@@_rectanglecolor [ #1 ] { #2 }
5868   { 1 - 1 }
5869   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5870 }

```

```

5871 \keys_define:nn { nicematrix / rowcolors }
5872 {
5873   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5874   cols .tl_set:N = \l_@@_cols_tl ,
5875   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5876   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5877 }

```

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*.

```

5878 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5879 {

```

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

```

5880 \pgfpicture
5881 \pgf@relevantforpicturesizefalse
5882 \seq_clear_new:N \l_@@_colors_seq
5883 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5884 \tl_clear_new:N \l_@@_cols_tl
5885 \tl_set:Nn \l_@@_cols_tl { - }
5886 \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).

```

5887 \int_zero_new:N \l_@@_color_int
5888 \int_set:Nn \l_@@_color_int 1
5889 \bool_if:NT \l_@@_respect_blocks_bool
5890 {

```

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`).

```

5891 % modified 2026-02-18
5892 \seq_set_filter:Nnn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5893 { \@@_not_in_exterior_p:nnnnn ##1 }
5894 }

```

`#2` is the list of intervals of rows.

```

5895 \clist_map_inline:nn { #2 }
5896 {
5897 \tl_set:Nn \l_tmpa_tl { ##1 }
5898 \tl_if_in:NnTF \l_tmpa_tl { - }
5899 { \@@_cut_on_hyphen:w ##1 \q_stop }
5900 { \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.

```

5901 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5902 \int_set:Nn \l_@@_color_int
5903 { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5904 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5905 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5906 {

```

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

```

5907 \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).

```

5908 \bool_if:NT \l_@@_respect_blocks_bool
5909 {
5910 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5911 { \@@_intersect_our_row_p:nnnnn #####1 }
5912 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5913 }
5914 \tl_set:Nn \l_@@_rows_tl
5915 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5916 \tl_set:Nn \l_@@_color_tl
5917 {
5918 \@@_color_index:n
5919 {
5920 \int_mod:nn
5921 { \l_@@_color_int - 1 }
5922 { \seq_count:N \l_@@_colors_seq }
5923 + 1
5924 }
5925 }

```

```

5926         \tl_if_empty:NF \l_@@_color_tl
5927         {
5928             \use:e
5929             {
5930                 \@@_add_to_colors_seq:nn
5931                 { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5932                 { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5933             }
5934         }
5935         \int_incr:N \l_@@_color_int
5936         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5937     }
5938 }
5939 \endpgfpicture
5940 }

```

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.

```

5941 \cs_new:Npn \@@_color_index:n #1
5942 {

```

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

```

5943     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5944     { \@@_color_index:n { #1 - 1 } }
5945     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5946 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5947 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5948 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5949 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5950 {
5951     \int_compare:nNnT { #3 } > \l_tmpb_int
5952     { \int_set:Nn \l_tmpb_int { #3 } }
5953 }

5954 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5955 {
5956     \int_if_zero:nTF { #4 }
5957     \prg_return_false:
5958     {
5959         \int_compare:nNnTF { #2 } > \c@jCol
5960         \prg_return_false:
5961         \prg_return_true:
5962     }
5963 }

```

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

```

5964 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5965 {
5966     \int_compare:nNnTF { #1 } > \l_tmpa_int
5967     \prg_return_false:
5968     {
5969         \int_compare:nNnTF \l_tmpa_int > { #3 }
5970         \prg_return_false:
5971         \prg_return_true:
5972     }
5973 }

```

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`).

```

5974 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5975 {
5976   \dim_compare:nNnTF { #1 } = \c_zero_dim
5977   {
5978     \bool_if:NTF \l_@@_nocolor_used_bool
5979     { \@@_cartesian_path_normal_ii: }
5980     {
5981       \clist_if_empty:NTF \l_@@_corners_cells_clist
5982       { \@@_cartesian_path_normal_i:n { #1 } }
5983       { \@@_cartesian_path_normal_ii: }
5984     }
5985   }
5986   { \@@_cartesian_path_normal_i:n { #1 } }
5987 }

```

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.

```

5988 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5989 {
5990   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5991   \clist_map_inline:Nn \l_@@_cols_tl
5992   {

```

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

```

5993     \def \l_tmpa_tl { ##1 }
5994     \tl_if_in:NnTF \l_tmpa_tl { - }
5995     { \@@_cut_on_hyphen:w ##1 \q_stop }
5996     { \def \l_tmpb_tl { ##1 } }
5997     \tl_if_empty:NTF \l_tmpa_tl
5998     { \def \l_tmpa_tl { 1 } }
5999     {
6000       \str_if_eq:eeT \l_tmpa_tl { * }
6001       { \def \l_tmpa_tl { 1 } }
6002     }
6003     \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6004     { \@@_error:n { Invalid~col~number } }
6005     \tl_if_empty:NTF \l_tmpb_tl
6006     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6007     {
6008       \str_if_eq:eeT \l_tmpb_tl { * }
6009       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6010     }
6011     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6012     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6013     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6014     \@@_qpoint:n { col - \l_tmpa_tl }
6015     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6016     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6017     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6018     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
6019     \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.

```

6020 \clist_map_inline:Nn \l_@@_rows_tl
6021 {
6022   \def \l_tmpa_tl { #####1 }
6023   \tl_if_in:NnTF \l_tmpa_tl { - }
6024     { \@@_cut_on_hyphen:w #####1 \q_stop }
6025     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6026   \tl_if_empty:NTF \l_tmpa_tl
6027     { \def \l_tmpa_tl { 1 } }
6028     {
6029       \str_if_eq:eeT \l_tmpa_tl { * }
6030       { \def \l_tmpa_tl { 1 } }
6031     }
6032   \tl_if_empty:NTF \l_tmpb_tl
6033     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6034     {
6035       \str_if_eq:eeT \l_tmpb_tl { * }
6036       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6037     }
6038   \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6039     { \@@_error:n { Invalid-row-number } }
6040   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6041     { \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`.

```

6042 \cs_if_exist:cF
6043 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6044 {
6045   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6046   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6047   \@@_qpoint:n { row - \l_tmpa_tl }
6048   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6049   \pgfpathrectanglecorners
6050     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6051     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6052   }
6053 }
6054 }
6055 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6056 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6057 {
6058   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6059   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6060 \clist_map_inline:Nn \l_@@_cols_tl
6061 {
6062   \@@_qpoint:n { col - ##1 }
6063   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6064     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6065     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6066   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6067   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6068 \clist_map_inline:Nn \l_@@_rows_tl
6069 {
6070   \@@_if_in_corner:nF { #####1 - ##1 }
6071   {
6072     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6073     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6074     \@@_qpoint:n { row - #####1 }

```

```

6075         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6076         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6077         {
6078             \pgfpathrectanglecorners
6079             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6080             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6081         }
6082     }
6083 }
6084 }
6085 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor` and `\@@_rowcolor:n` (used in `\@@_rowcolor`).

```

6086 \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

```

6087 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6088 {
6089     \bool_set_true:N \l_@@_nocolor_used_bool
6090     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6091     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6092     \clist_map_inline:Nn \l_@@_rows_tl
6093     {
6094         \clist_map_inline:Nn \l_@@_cols_tl
6095         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6096     }
6097 }

```

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.

```

6098 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6099 {
6100     \clist_set_eq:NN \l_tmpa_clist #1
6101     \clist_clear:N #1
6102     \clist_map_inline:Nn \l_tmpa_clist
6103     {

```

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

```

6104         \def \l_tmpa_tl { ##1 }
6105         \tl_if_in:NnTF \l_tmpa_tl { - }
6106         { \@@_cut_on_hyphen:w ##1 \q_stop }
6107         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6108         \bool_lazy_or:nnT
6109         { \str_if_eq_p:ee \l_tmpa_tl { * } }
6110         { \tl_if_blank_p:o \l_tmpa_tl }
6111         { \def \l_tmpa_tl { 1 } }
6112         \bool_lazy_or:nnT
6113         { \str_if_eq_p:ee \l_tmpb_tl { * } }
6114         { \tl_if_blank_p:o \l_tmpb_tl }
6115         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6116         \int_compare:nNnT \l_tmpb_tl > { #2 }
6117         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6118         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6119         { \clist_put_right:Nn #1 { #####1 } }
6120     }
6121 }

```

The following command will be linked to `\cellcolor` in the tabular.

```
6122 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6123 {
6124   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6125   {
```

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`).

```
6126     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6127     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6128   }
6129   \ignorespaces
6130 }

6131 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6132 { \@@_error:n { cellcolor~in~Block } }
6133 % \end{macrocode}
6134 %
6135 % \begin{macrocode}
6136 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6137 { \@@_error:n { rowcolor~in~Block } }
6138 % \end{macrocode}
6139 %
6140 % \bigskip
6141 % The following command will be linked to |\rowcolor| in the tabular.
6142 % \begin{macrocode}
6143 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6144 {
6145   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6146   {
6147     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6148     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6149     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6150   }
6151   \ignorespaces
6152 }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
6153 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6154 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
6155 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6156 {
```

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`.

```
6157   \seq_gclear:N \g_tmpa_seq
6158   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6159   { \@@_rowlistcolors_tabular:nnnn #1 }
6160   \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`.

```
6161   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6162   {
```

```

6163     { \int_use:N \c@iRow }
6164     { \exp_not:n { #1 } }
6165     { \exp_not:n { #2 } }
6166     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6167   }
6168   \ignorespaces
6169 }

```

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`).

```

6170 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6171 {
6172   \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).

```

6173   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6174   {
6175     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6176     {
6177       \@@_rowlistcolors
6178       [ \exp_not:n { #2 } ]
6179       { #1 - \int_eval:n { \c@iRow - 1 } }
6180       { \exp_not:n { #3 } }
6181       [ \exp_not:n { #4 } ]
6182     }
6183   }
6184 }

```

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.

```

6185 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6186 {
6187   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6188   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6189   \seq_gclear:N \g_@@_rowlistcolors_seq
6190 }

6191 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6192 {
6193   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6194   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6195 }

```

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.

```

6196 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6197 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

6198   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
6199   {

```

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).

```

6200     \tl_gput_left:N\g_@@_pre_code_before_tl
6201     {
6202         \exp_not:N \columncolor [ #1 ]
6203         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6204     }
6205 }
6206 }

6207 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6208 {
6209     \clist_map_inline:nn { #1 }
6210     {
6211         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6212         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6213         \columncolor { nocolor } { ##1 }
6214     }
6215 }

6216 \cs_new_protected:Npn \@@_EmptyRow:n #1
6217 {
6218     \clist_map_inline:nn { #1 }
6219     {
6220         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6221         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6222         \rowcolor { nocolor } { ##1 }
6223     }
6224 }

```

## 22 The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

6225 \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`.

```

6226 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6227 {
6228     \int_if_zero:nTF \l_@@_first_col_int
6229     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6230     {
6231         \int_if_zero:nTF \c@jCol
6232         {
6233             \int_compare:nNnF \c@iRow = { -1 }
6234             {

```

```

6235         \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6236         { #1 }
6237     }
6238 }
6239 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6240 }
6241 }

```

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.

```

6242 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6243 {
6244     \int_if_zero:nF \c@iRow
6245     {
6246         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6247         {
6248             \int_compare:nNnT \c@jCol > \c_zero_int
6249             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6250         }
6251     }
6252 }

```

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`.

```

6253 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6254 {
6255     \IfPackageLoadedTF { tikz }
6256     {
6257         \IfPackageLoadedTF { booktabs }
6258         { #2 }
6259         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6260     }
6261     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6262 }
6263 \NewExpandableDocumentCommand { \@@_TopRule } { }
6264 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6265 \cs_new:Npn \@@_TopRule_i:
6266 {
6267     \noalign \bgroup
6268     \peek_meaning:NTF [
6269     { \@@_TopRule_ii: }
6270     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6271 }
6272 \NewDocumentCommand \@@_TopRule_ii: { o }
6273 {
6274     \tl_gput_right:Ne \g_@@_rules_tl
6275     {
6276         \@@_draw_hrule:n
6277         {
6278             position = \int_eval:n { \c@iRow + 1 } ,
6279             tikz =
6280             {
6281                 line-width = #1 ,
6282                 yshift = 0.25 \arrayrulewidth ,
6283                 shorten~< = - 0.5 \arrayrulewidth
6284             } ,
6285             total-width = #1
6286         }

```

```

6287     }
6288     \skip_vertical:n { \belowrulesep + #1 }
6289     \egroup
6290 }

6291 \NewExpandableDocumentCommand { \@@_BottomRule } { } {
6292   { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6293   \cs_new:Npn \@@_BottomRule_i:
6294     {
6295       \noalign \bgroup
6296       \peek_meaning:NTF [
6297         { \@@_BottomRule_ii: }
6298         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6299       }

6300   \NewDocumentCommand \@@_BottomRule_ii: { o }
6301   {
6302     \tl_gput_right:Ne \g_@@_rules_tl
6303     {
6304       \@@_draw_hrulerule:n
6305       {
6306         position = \int_eval:n { \c@iRow + 1 } ,
6307         tikz =
6308         {
6309           line-width = #1 ,
6310           yshift = 0.25 \arrayrulewidth ,
6311           shorten-< = - 0.5 \arrayrulewidth
6312         } ,
6313         total-width = #1 ,
6314       }
6315     }
6316     \skip_vertical:N \aboverulesep
6317     \@@_create_row_node_i:
6318     \skip_vertical:n { #1 }
6319     \egroup
6320   }

6321 \NewExpandableDocumentCommand { \@@_MidRule } { } {
6322   { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6323   \cs_new:Npn \@@_MidRule_i:
6324     {
6325       \noalign \bgroup
6326       \peek_meaning:NTF [
6327         { \@@_MidRule_ii: }
6328         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6329       }

6330   \NewDocumentCommand \@@_MidRule_ii: { o }
6331   {
6332     \skip_vertical:N \aboverulesep
6333     \@@_create_row_node_i:
6334     \tl_gput_right:Ne \g_@@_rules_tl
6335     {
6336       \@@_draw_hrulerule:n
6337       {
6338         position = \int_eval:n { \c@iRow + 1 } ,
6339         tikz =
6340         {
6341           line-width = #1 ,
6342           yshift = 0.25 \arrayrulewidth ,
6343           shorten-< = - 0.5 \arrayrulewidth
6344         } ,
6345         total-width = #1 ,
6346       }
6347     }

```

```

6348     \skip_vertical:n { \belowrulesep + #1 }
6349     \egroup
6350 }

```

## General system for drawing rules

```

6351 \AtBeginDocument
6352 {
6353     \cs_new_protected:Npe \@@_draw_rules:
6354     {
6355         \c_@@_pgfortikzpicture_tl
6356         \@@_draw_rules_i:
6357         \c_@@_endpgfortikzpicture_tl
6358     }
6359 }
6360 \cs_new_protected:Npn \@@_draw_rules_i:
6361 {
6362     \bool_lazy_all:nT
6363     {
6364         { \clist_if_empty_p:N \l_@@_corners_clist }
6365         { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6366         { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6367         { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6368     }
6369     {
6370         \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6371         \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6372     }
6373     \pgfrememberpicturepositiononpagetrue
6374     \pgf@relevantforpicturesizefalse
6375     \pgfsetrectcap
6376     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6377     \CT@arc@
6378     \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6379     \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6380     \g_@@_rules_tl
6381 }

```

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”.

```

6382 \keys_define:nn { nicematrix / rules-after }
6383 {
6384     position .int_set:N = \l_@@_position_int ,
6385     start .int_set:N = \l_@@_start_int ,
6386     start .initial:n = 1 ,
6387     end .int_set:N = \l_@@_end_int ,
6388     multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6389     dotted .code:n =
6390         \bool_set_true:N \l_@@_dotted_bool
6391         \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6392         \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6393     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]`.

```

6394     color .code:n =
6395         \@@_set_CTarc:n { #1 }

```

```

6396     \CT@arc@
6397     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6398     color .value_required:n = true ,
6399     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6400     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
  }
}

6401 total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6402 unknown .code:n =
6403   \@@_unknown_key:nn
6404     { nicematrix / rules-after }
6405     { Unknown-key-for-a-rule } ,
6406   tikz .value_required:n = true
6407 }

```

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.

```

6408 \AtBeginDocument
6409 {
6410   \IfPackageLoadedTF { tikz }
6411   {
6412     \keys_define:nn { nicematrix / rules-after }
6413     {
6414       tikz .code:n =
6415         \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6416         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6417         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
6418     }
6419   }
6420   {
6421     \keys_define:nn { nicematrix / rules-after }
6422     { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6423   }
6424 }

6425 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6426 {
6427   \int_set:Nn \l_@@_multiplicity_int { #1 }
6428   \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6429   \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6430 }

```

## 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.

```

6431 \cs_new_protected:Npn \@@_draw_vrule:n #1
6432 {

```

The group is for the options.

```

6433 {
6434   \int_set_eq:NN \l_@@_end_int \c@iRow
6435   \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).

```

6436   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6437   \socket_use:n { nicematrix / draw-vrule }
6438 }
6439 }

```

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.

```

6440 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6441 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6442 {
6443   \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`.

```

6444   \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).

```

6445   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6446   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6447   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6448   \l_tmpa_tl
6449   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6450   \@@_test_v:
6451   \bool_if:NTF \g_tmpa_bool
6452   {
6453     \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.

```

6454     { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6455     { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6456   }
6457   {
6458     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6459     {
6460       \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6461       \socket_use:n { nicematrix / vsegment }
6462       \int_zero:N \l_@@_segment_start_int
6463     }
6464   }
6465 }
6466 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int

```

```

6467     {
6468         \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6469         \socket_use:n { nicematrix / vsegment }
6470     }
6471     \group_end:
6472 }
6473 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6474 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6475 {
6476     \group_begin:
6477     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6478     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6479     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6480     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6481     \socket_use:n { nicematrix / vsegment }
6482     \group_end:
6483 }

```

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.

```

6484 \cs_new_protected:Npn \@@_test_v:
6485 {
6486     \bool_gset_true:N \g_tmpa_bool
6487     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6488     \bool_if:NT \g_tmpa_bool
6489     {
6490         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6491         { \@@_test_vline_in_block:nnnnn ##1 }
6492         \bool_if:NT \g_tmpa_bool
6493         {
6494             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6495             { \@@_test_vline_in_block:nnnnn ##1 }
6496             \bool_if:NT \g_tmpa_bool
6497             {
6498                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6499                 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6500             }
6501         }
6502     }
6503 }
6504 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6505 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6506 {
6507     {
6508         \int_compare:nNnTF \l_@@_position_int > \c@jCol
6509         { \bool_set_false:N \l_tmpa_bool }
6510         {
6511             \@@_test_h:
6512             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6513         }
6514         \int_compare:nNnTF \l_@@_position_int = 1
6515         { \bool_gset_false:N \g_tmpa_bool }
6516         {
6517             \tl_set:N \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6518             \@@_test_h:
6519         }
6520         \bool_gset:Nn \g_tmpa_bool
6521         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6522     }
6523     \bool_if:NT \g_tmpa_bool

```

```

6524     {
6525         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6526         \socket_use:n { nicematrix / vsegment }
6527         \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6528     }
6529 }

6530 \cs_new_protected:Npn \@@_test_in_corner_v:
6531 {
6532     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6533     {
6534         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6535         { \bool_set_false:N \g_tmpa_bool }
6536     }
6537     {
6538         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6539         {
6540             \int_compare:nNnTF \l_tmpb_tl = 1
6541             { \bool_set_false:N \g_tmpa_bool }
6542             {
6543                 \@@_if_in_corner:nT
6544                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6545                 { \bool_set_false:N \g_tmpa_bool }
6546             }
6547         }
6548     }
6549 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standar` 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.

```

6550 \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`.

```

6551 \socket_new_plugin:nnn { nicematrix / vsegment } { standard }
6552 {
6553     \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6554     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6555     \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6556     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6557     \pgfusepathqstroke
6558 }

6559 \socket_assign_plugin:nn { nicematrix / vsegment } { standard }

6560 \socket_new_plugin:nnn { nicematrix / vsegment } { multiple }
6561 {
6562     \dim_add:Nn \l_@@_x_initial_dim
6563     {
6564         - 0.5 \l_@@_rule_width_after_dim
6565         +
6566         ( \arrayrulewidth * \l_@@_multiplicity_int
6567           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6568     }
6569     \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6570     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y

```

```

6571 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6572 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6573 \bool_lazy_and:nnT
6574 { \cs_if_exist_p:N \CT@drsc@ }
6575 { ! \tl_if_blank_p:o \CT@drsc@ }
6576 {
6577   {
6578     \CT@drsc@
6579     \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6580     \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6581     \dim_set:Nn \l_@@_tmpd_dim
6582       {
6583         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6584         * ( \l_@@_multiplicity_int - 1 )
6585       }
6586     \pgfpathrectanglecorners
6587       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6588       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6589     \pgfusepath { fill }
6590   }
6591 }
6592 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6593 \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6594 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6595 {
6596   \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6597   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6598   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6599 }
6600 \pgfusepathqstroke
6601 }

6602 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6603 {
6604   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6605   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6606   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6607   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6608   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6609   \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`

```

6610 \@@_draw_line:
6611 }

```

```

6612 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6613 {
6614   {

```

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).

```

6615 \tl_if_empty:NF \l_@@_rule_color_tl
6616 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6617 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6618 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6619 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6620 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6621 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6622 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6623 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6624 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6625 }
6626 }

```

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).

```

6627 \cs_new_protected:Npn \@@_draw_key_vlines:
6628 {
6629 {
6630 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6631 \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currying in the following code.

```

6632 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6633 { \@@_lines_step_inline:n \c@jCol }
6634 { \clist_map_inline:Nn \l_@@_vlines_clist }
6635 {
6636 \int_set:Nn \l_@@_position_int { ##1 }
6637 \socket_use:n { nicematrix / draw-vrule }
6638 }
6639 }
6640 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6641 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6642 {
6643 \int_step_inline:nnn
6644 { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6645 {
6646 \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6647 { #1 }
6648 { \int_eval:n { #1 + 1 } }
6649 }
6650 }

```

## The horizontal rules

`\@@_draw_hruler: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.

```

6651 \cs_new_protected:Npn \@@_draw_hruler:n #1
6652 {
6653 {
6654 \int_set_eq:NN \l_@@_end_int \c@jCol
6655 \keys_set_known:nn { nicematrix / rules-after } { #1 }

```

```

6656     \socket_use:nn { nicematrix / draw-hrule }
6657   }
6658 }

```

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.

```

6659 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6660 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6661 {
6662   \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`.

```

6663   \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).

```

6664   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6665   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6666   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6667   \l_tmpb_tl
6668   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6669     \@@_test_h:
6670     \bool_if:NTF \g_tmpa_bool
6671     {
6672       \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.

```

6673       { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6674       { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6675     }
6676   {
6677     \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6678     {
6679       \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

6680     \socket_use:n { nicematrix / hsegment }
6681     \int_zero:N \l_@@_segment_start_int
6682   }
6683 }
6684 }
6685 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6686 {
6687   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6688   \socket_use:n { nicematrix / hsegment }
6689 }
6690 \group_end:
6691 }

```

```

6692 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6693 \socket_new_plug:nnn { nicematrix / draw-hrule} { quick }
6694 {
6695   \group_begin:
6696   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6697   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6698   \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6699   \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6700   \socket_use:n { nicematrix / hsegment }
6701   \group_end:
6702 }

```

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.

```

6703 \cs_new_protected:Npn \@@_test_h:
6704 {
6705   \bool_gset_true:N \g_tmpa_bool
6706   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6707   \bool_if:NT \g_tmpa_bool
6708   {
6709     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6710     { \@@_test_hline_in_block:nnnnn ##1 }
6711     \bool_if:NT \g_tmpa_bool
6712     {
6713       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6714       { \@@_test_hline_in_block:nnnnn ##1 }
6715       \bool_if:NT \g_tmpa_bool
6716       {
6717         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6718         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6719       }
6720     }
6721   }
6722 }

6723 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }
6724 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6725 {
6726   {
6727     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6728     { \bool_set_false:N \l_tmpa_bool }
6729     {
6730       \@@_test_v:
6731       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6732     }
6733     \int_compare:nNnTF \l_@@_position_int = 1
6734     { \bool_gset_false:N \g_tmpa_bool }
6735     {
6736       \tl_set:N \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6737       \@@_test_v:
6738     }
6739     \bool_gset:Nn \g_tmpa_bool
6740     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6741   }
6742   \bool_if:NT \g_tmpa_bool
6743   {
6744     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6745     \socket_use:n { nicematrix / hsegment }
6746     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6747   }
6748 }

```

```

6749 \cs_new_protected:Npn \@@_test_in_corner_h:
6750 {
6751   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6752   {
6753     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6754     { \bool_set_false:N \g_tmpa_bool }
6755   }
6756   {
6757     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6758     {
6759       \int_compare:nNnTF \l_tmpa_tl = 1
6760       { \bool_set_false:N \g_tmpa_bool }
6761       {
6762         \@@_if_in_corner:nT
6763         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6764         { \bool_set_false:N \g_tmpa_bool }
6765       }
6766     }
6767   }
6768 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug `standar` for simple rules;
- a plug `multiplity` 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.

```

6769 \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`.

```

6770 \socket_new_plug:nnn { nicematrix / hsegment } { standard }
6771 {
6772   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6773   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6774   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6775   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6776   \pgfusepathqstroke
6777 }
6778 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
6779 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6780 {
6781   \dim_add:Nn \l_@@_y_initial_dim
6782   {
6783     - 0.5 \l_@@_rule_width_after_dim
6784     +
6785     ( \arrayrulewidth * \l_@@_multiplicity_int
6786       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6787   }
6788   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6789   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6790   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6791   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6792   \bool_lazy_and:nnT
6793   { \cs_if_exist_p:N \CT@drsc@ }
6794   { ! \tl_if_blank_p:o \CT@drsc@ }
6795   {
6796     {

```

```

6797 \CT@drsc@
6798 \dim_set:Nn \l_@@_tmpd_dim
6799 {
6800   \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6801   * ( \l_@@_multiplicity_int - 1 )
6802 }
6803 \pgfpathrectanglecorners
6804 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6805 { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6806 \pgfusepathqfill
6807 }
6808 }
6809 \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6810 \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6811 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6812 {
6813   \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6814   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6815   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6816 }
6817 \pgfusepathqstroke
6818 }

```

Now, the case of a dotted line.

```
\begin{bNiceMatrix}
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
```

```
1 & 2 & 3 & 4 \\\
```

```
\hline
```

```
1 & 2 & 3 & 4 \\\
```

```
\hdottedline
```

```
1 & 2 & 3 & 4
```

```
\end{bNiceMatrix}
```

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

```

6819 \socket_new_plugin:nnn { nicematrix / hsegment } { dotted }
6820 {
6821   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6822   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6823   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6824   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6825   \int_compare:nNnT \l_@@_segment_start_int = 1
6826   {
6827     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6828     \bool_if:NF \g_@@_delims_bool
6829     { \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.

```

6830 \tl_if_eq:NnF \g_@@_left_delim_tl (
6831   { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6832 )
6833 \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6834 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6835 \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6836 {
6837   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6838   \bool_if:NF \g_@@_delims_bool

```

```

6839         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6840         \tl_if_eq:NnF \g_@@_right_delim_tl )
6841         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6842     }

```

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`

```

6843     \@@_draw_line:
6844 }

```

```

6845 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6846 {
6847     {

```

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).

```

6848     \tl_if_empty:NF \l_@@_rule_color_tl
6849     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6850     \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6851     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6852     \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6853     \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6854     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6855     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6856     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6857     -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6858 }
6859 }

```

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).

```

6860 \cs_new_protected:Npn \@@_draw_key_hlines:
6861 {
6862     {
6863         \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6864         \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6865     \str_if_eq:eeTF \l_@@_hlines_clist { all }
6866     { \@@_lines_step_inline:n \c@iRow }
6867     { \clist_map_inline:Nn \l_@@_hlines_clist }
6868     {
6869         \int_set:Nn \l_@@_position_int { ##1 }
6870         \socket_use:n { nicematrix / draw-hrule }
6871     }
6872 }
6873 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6874 \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.

```

6875 \cs_set:Npn \@@_Hline_i:n #1
6876 {
6877   \peek_remove_spaces:n
6878   {
6879     \peek_meaning:NTF \Hline
6880     { \@@_Hline_ii:nn { #1 + 1 } }
6881     { \@@_Hline_iii:n { #1 } }
6882   }
6883 }
6884 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6885 \cs_set:Npn \@@_Hline_iii:n #1
6886 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6887 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6888 {
6889   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6890   \skip_vertical:N \l_@@_rule_width_before_dim
6891   \tl_gput_right:Ne \g_@@_rules_tl
6892   {

```

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:

```

6893 {
6894   \int_compare:nNnT { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6895   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6896   \dim_set:Nn \l_@@_rule_width_after_dim
6897   { \dim_use:N \l_@@_rule_width_before_dim }
6898   \@@_draw_hrule:n { #2 }
6899 }
6900 }
6901 \egroup
6902 }

```

## 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`).

```

6903 \cs_new_protected:Npn \@@_custom_line:n #1
6904 {
6905   \str_clear:N \l_@@_letter_str
6906   \str_clear:N \l_@@_command_str
6907   \str_clear:N \l_@@_ccommand_str
6908   \tl_clear_new:N \l_@@_other_keys_tl
6909   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6910   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6911   {
6912     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6913     \str_set:Ne \l_@@_command_str
6914     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6915   }
6916   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6917   {
6918     \str_set:Ne \l_@@_ccommand_str
6919     { \str_tail:N \l_@@_ccommand_str }
6920     \str_set:Ne \l_@@_ccommand_str
6921     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6922   }

```

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.

```

6923   \bool_lazy_all:nTF
6924   {
6925     { \str_if_empty_p:N \l_@@_letter_str }
6926     { \str_if_empty_p:N \l_@@_command_str }
6927     { \str_if_empty_p:N \l_@@_ccommand_str }
6928   }
6929   { \@@_error:n { No~letter~and~no~command } }
6930   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6931 }
6932 \keys_define:nn { nicematrix / custom-line }
6933 {
6934   letter .str_set:N = \l_@@_letter_str ,
6935   letter .value_required:n = true ,
6936   command .str_set:N = \l_@@_command_str ,
6937   command .value_required:n = true ,
6938   ccommand .str_set:N = \l_@@_ccommand_str ,
6939   ccommand .value_required:n = true
6940 }

```

```

6941 \cs_new_protected:Npn \@@_custom_line_i:n #1
6942 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6943   \bool_set_false:N \l_@@_tikz_rule_bool
6944   \bool_set_false:N \l_@@_dotted_rule_bool
6945   \bool_set_false:N \l_@@_color_bool
6946   \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6947   \bool_if:NT \l_@@_tikz_rule_bool
6948   {
6949     \IfPackageLoadedF { tikz }
6950     { \@@_error:n { tikz~in~custom~line~without~tikz } }
6951     \bool_if:NT \l_@@_color_bool
6952     { \@@_error:n { color~in~custom~line~with~tikz } }
6953   }
6954   \bool_if:NT \l_@@_dotted_rule_bool
6955   {
6956     \int_compare:nNnT \l_@@_multiplicity_int > 1
6957     { \@@_error:n { key~multiplicity~with~dotted } }
6958   }
6959   \str_if_empty:NF \l_@@_letter_str
6960   {
6961     \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6962     { \@@_error:n { Several~letters } }
6963     {
6964       \tl_if_in:NoTF

```

```

6965         \c_@@_forbidden_letters_str
6966         \l_@@_letter_str
6967         { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6968     }

```

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.

```

6969         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6970         { \@@_v_custom_line:nn { #1 } }
6971     }
6972 }
6973 }
6974 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6975 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6976 }
6977 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6978 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6979 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/rules-after}`). That's why the following set of keys has some keys which are no-op.

```

6980 \keys_define:nn { nicematrix / custom-line-bis }
6981 {
6982     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6983     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6984     color .value_required:n = true ,
6985     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6986     tikz .value_required:n = true ,
6987     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6988     dotted .value_forbidden:n = true ,
6989     total-width .code:n = { } ,
6990     total-width .value_required:n = true ,
6991     sep-color .code:n = { } ,
6992     sep-color .value_required:n = true ,
6993     unknown .code:n =
6994         \@@_unknown_key:nn
6995         { nicematrix / custom-line-bis }
6996         { Unknown-key~for~custom-line }
6997 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6998 \bool_new:N \l_@@_dotted_rule_bool
6999 \bool_new:N \l_@@_tikz_rule_bool
7000 \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).

```

7001 \keys_define:nn { nicematrix / custom-line-width }
7002 {
7003     multiplicity .int_set:N = \l_@@_tmpc_int ,
7004     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7005     total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7006         \bool_set_true:N \l_@@_total_width_bool ,
7007     total-width .value_required:n = true ,
7008     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7009 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. #1 is the whole set of keys to pass to the command \@@\_draw\_hrule:n (which is in the internal \CodeAfter).

```
7010 \cs_new_protected:Npn \@@_h_custom_line:n #1
7011 {
```

We use \cs\_set:nopar: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).

```
7012 \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7013 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7014 }
```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter c as in \cline). #1 is the whole set of keys to pass to the command \@@\_draw\_hrule:n (which is in the internal \CodeAfter).

```
7015 \cs_new_protected:Npn \@@_c_custom_line:n #1
7016 {
```

Here, we need an expandable command since it begins with an \noalign.

```
7017 \exp_args:Nc \DeclareExpandableDocumentCommand
7018 { nicematrix - \l_@@_ccommand_str }
7019 { 0 { } m }
7020 {
7021 \noalign
7022 {
7023 \@@_compute_rule_width:n { #1 , ##1 }
7024 \skip_vertical:n \l_@@_rule_width_before_dim
7025 \clist_map_inline:nn
7026 { ##2 }
7027 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7028 }
7029 }
7030 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7031 }
```

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*.

```
7032 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7033 {
7034 \tl_if_in:nnTF { #2 } { - }
7035 { \@@_cut_on_hyphen:w #2 \q_stop }
7036 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7037 \tl_gput_right:Ne \g_@@_rules_tl
7038 {
7039 \@@_draw_hrule:n
7040 {
7041 #1 ,
7042 start = \l_tmpa_tl ,
7043 end = \l_tmpb_tl ,
7044 position = \int_eval:n { \c@iRow + 1 } ,
7045 total-width = \dim_use:N \l_@@_rule_width_before_dim
7046 }
7047 }
7048 }

7049 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7050 {
7051 \bool_set_false:N \l_@@_tikz_rule_bool
7052 \bool_set_false:N \l_@@_total_width_bool
7053 \bool_set_false:N \l_@@_dotted_rule_bool
7054 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7055 \bool_if:NF \l_@@_total_width_bool
```

```

7056 {
7057   \bool_if:NTF \l_@@_dotted_rule_bool
7058   { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7059   {
7060     \bool_if:NF \l_@@_tikz_rule_bool
7061     {
7062       \dim_set:Nn \l_@@_rule_width_before_dim
7063       {
7064         \arrayrulewidth * \l_@@_tmpc_int
7065         + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7066       }
7067     }
7068   }
7069 }
7070 }

```

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

```

7071 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7072 {
7073   \str_if_eq:nnTF { #2 } { [ ]
7074     { \@@_v_custom_line_i:nw { #1 } [ ]
7075       { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7076     }
7077   \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7078   { \@@_v_custom_line:nn { #1 , #2 } }
7079   \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7080   {
7081     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7082   \tl_gput_right:Ne \g_@@_array_preamble_tl
7083   {
7084     \exp_not:N !
7085     { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7086   }
7087   \tl_gput_right:Ne \g_@@_rules_tl
7088   {

```

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.

```

7089 {
7090   \dim_set:Nn \l_@@_rule_width_after_dim
7091   { \dim_use:N \l_@@_rule_width_before_dim }
7092   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7093   \@@_draw_vrule:n { #2 }
7094 }
7095 }
7096 \@@_rec_preamble:n #1
7097 }
7098 \@@_custom_line:n
7099 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }
7100 \AtBeginDocument
7101 {
7102   \IfPackageLoadedT { tikz }

```

```

7103 {
7104   \cs_if_exist:cTF { tikz@library@decorations@loaded }
7105   {
7106     \@@_custom_line:n
7107     {
7108       letter = ; ,
7109       command = \hdashedline ,
7110       ccommand = \cdashedline ,
7111       tikz =
7112       {
7113         dash-pattern = on-4~pt-off-2pt,
7114         dash-expand-off ,
7115         line-cap = butt
7116       } ,
7117       total-width = \pgflinewidth
7118     }
7119   }
7120   {
7121     \@@_custom_line:n
7122     {
7123       letter = ; ,
7124       command = \hdashedline ,
7125       ccommand = \cdashedline ,
7126       tikz = { dash-pattern = on-4~pt-off-2pt , line-cap = butt } ,
7127       total-width = \pgflinewidth
7128     }
7129   }
7130 }
7131 }

```

## The key default-line

```

7132 \keys_define:nn { nicematrix / default-line }
7133 {
7134   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7135   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7136   color .value_required:n = true ,
7137   tikz .code:n =
7138     \bool_set_true:N \l_@@_tikz_rule_bool
7139     \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7140   tikz .value_required:n = true ,
7141   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7142   dotted .value_forbidden:n = true ,
7143   total-width .code:n = { } ,
7144   total-width .value_required:n = true ,
7145   sep-color .code:n = { } ,
7146   sep-color .value_required:n = true ,
7147   unknown .code:n =
7148     \@@_unknown_key:nn
7149     { nicematrix / default-line }
7150     { Unknown-key-for-default-line }
7151 }
7152 \cs_new_protected:Npn \@@_default_line:n #1
7153 {
7154   \bool_set_false:N \l_@@_tikz_rule_bool
7155   \bool_set_false:N \l_@@_dotted_rule_bool
7156   \bool_set_false:N \l_@@_color_bool
7157   \keys_set:nn { nicematrix / default-line } { #1 }
7158   \bool_if:NT \l_@@_tikz_rule_bool
7159   {
7160     \IfPackageLoadedF { tikz }
7161     { \@@_error:n { tikz-in-custom-line-without-tikz } }
7162     \bool_if:NT \l_@@_color_bool

```

```

7163         { \@@_error:n { color-in-custom-line-with-tikz } }
7164     }
7165     \bool_if:NT \l_@@_dotted_rule_bool
7166     {
7167         \int_compare:nNnT \l_@@_multiplicity_int > 1
7168         { \@@_error:n { key~multiplicity~with~dotted } }
7169     }
7170     \bool_if:NTF \l_@@_tikz_rule_bool
7171     {
7172         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7173         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7174     }
7175     {
7176         \bool_if:NTF \l_@@_dotted_rule_bool
7177         {
7178             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7179             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7180         }
7181         {
7182             \bool_if:NTF \l_@@_multiple_rule_bool
7183             {
7184                 \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7185                 \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7186             }
7187         }
7188     }
7189 }

```

### 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.

```

7190 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7191 {
7192     \int_compare:nNnT \l_tmpa_tl > { #1 }
7193     {
7194         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7195         {
7196             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7197             {
7198                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7199                 { \bool_gset_false:N \g_tmpa_bool }
7200             }
7201         }
7202     }
7203 }

```

The same for vertical rules.

```

7204 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7205 {
7206     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7207     {
7208         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7209         {
7210             \int_compare:nNnT \l_tmpb_tl > { #2 }
7211             {
7212                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7213                 { \bool_gset_false:N \g_tmpa_bool }
7214             }
7215         }
7216     }
7217 }

```

```

7218 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7219 {
7220   \int_compare:nNt \l_tmpb_tl > { #2 - 1 }
7221   {
7222     \int_compare:nNt \l_tmpb_tl < { #4 + 1 }
7223     {
7224       \int_compare:nNtF \l_tmpa_tl = { #1 }
7225       { \bool_gset_false:N \g_tmpa_bool }
7226       {
7227         \int_compare:nNt \l_tmpa_tl = { #3 + 1 }
7228         { \bool_gset_false:N \g_tmpa_bool }
7229       }
7230     }
7231   }
7232 }

7233 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7234 {
7235   \int_compare:nNt \l_tmpa_tl > { #1 - 1 }
7236   {
7237     \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
7238     {
7239       \int_compare:nNtF \l_tmpb_tl = { #2 }
7240       { \bool_gset_false:N \g_tmpa_bool }
7241       {
7242         \int_compare:nNt \l_tmpb_tl = { #4 + 1 }
7243         { \bool_gset_false:N \g_tmpa_bool }
7244       }
7245     }
7246   }
7247 }

```

## 23 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

7248 \cs_new_protected:Npn \@@_compute_corners:
7249 {
7250   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7251   { \@@_mark_cells_of_block:nnnnn ##1 }

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

7252   \clist_clear:N \l_@@_corners_cells_clist
7253   \clist_map_inline:Nn \l_@@_corners_clist
7254   {
7255     \str_case:nnF { ##1 }
7256     {
7257       { NW }
7258       { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7259       { NE }
7260       { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7261       { SW }
7262       { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7263       { SE }
7264       { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7265     }

```

```

7266         { \@@_error:nn { bad-corner } { ##1 } }
7267     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7268     \clist_if_empty:NF \l_@@_corners_cells_clist
7269     {

```

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.

```

7270         \tl_gput_right:Ne \g_@@_aux_tl
7271         {
7272             \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7273             { \l_@@_corners_cells_clist }
7274         }
7275     }
7276 }

```

```

7277 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7278 {
7279     \int_step_inline:nnn { #1 } { #3 }
7280     {
7281         \int_step_inline:nnn { #2 } { #4 }
7282         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7283     }
7284 }

```

```

7285 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7286 {
7287     \cs_if_exist:cTF
7288     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7289     \prg_return_true:
7290     \prg_return_false:
7291 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_clist`.

The six arguments of `\@@_compute_corner:nnnnnn` are as follow:

- `#1` and `#2` are the number of row and column of the cell which is actually in the corner;
- `#3` and `#4` are the steps in rows and the step in columns when moving from the corner;
- `#5` is the number of the final row when scanning the rows from the corner;
- `#6` is the number of the final column when scanning the columns from the corner.

```

7292 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7293 {

```

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.

```

7294     \bool_set_false:N \l_tmpa_bool
7295     \int_zero_new:N \l_@@_last_empty_row_int
7296     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7297     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7298     {
7299         \bool_lazy_or:nnTF
7300         {

```

```

7301         \cs_if_exist_p:c
7302         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7303     }
7304     { \@@_if_in_block_p:nn { ##1 } { #2 } }
7305     { \bool_set_true:N \l_tmpa_bool }
7306     {
7307         \bool_if:NF \l_tmpa_bool
7308         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7309     }
7310 }

```

Now, you determine the last empty cell in the row of number 1.

```

7311     \bool_set_false:N \l_tmpa_bool
7312     \int_zero_new:N \l_@@_last_empty_column_int
7313     \int_set:Nn \l_@@_last_empty_column_int { #2 }
7314     \int_step_inline:nnnn { #2 } { #4 } { #6 }
7315     {
7316         \bool_lazy_or:nnTF
7317         {
7318             \cs_if_exist_p:c
7319             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7320         }
7321         { \@@_if_in_block_p:nn { #1 } { ##1 } }
7322         { \bool_set_true:N \l_tmpa_bool }
7323         {
7324             \bool_if:NF \l_tmpa_bool
7325             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7326         }
7327     }

```

Now, we loop over the rows.

```

7328     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7329     {

```

We treat the row number ##1 with another loop.

```

7330         \bool_set_false:N \l_tmpa_bool
7331         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7332         {
7333             \bool_lazy_or:nnTF
7334             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7335             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7336             { \bool_set_true:N \l_tmpa_bool }
7337             {
7338                 \bool_if:NF \l_tmpa_bool
7339                 {
7340                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7341                     \clist_put_right:Nn
7342                     \l_@@_corners_cells_clist
7343                     { ##1 - #####1 }
7344                     \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7345                 }
7346             }
7347         }
7348     }
7349 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7350 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7351 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:  
`\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

## 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.

```
7352 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
7353 \keys_define:nn { nicematrix / NiceMatrixBlock }
7354 {
7355   auto-columns-width .code:n =
7356   {
7357     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7358     \dim_gzero_new:N \g_@@_max_cell_width_dim
7359     \bool_set_true:N \l_@@_auto_columns_width_bool
7360   }
7361 }

7362 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
7363 {
7364   \int_gincr:N \g_@@_NiceMatrixBlock_int
7365   \dim_zero:N \l_@@_columns_width_dim
7366   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7367   \bool_if:NT \l_@@_block_auto_columns_width_bool
7368   {
7369     \cs_if_exist:cT
7370     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7371     {
7372       \dim_set:Nn \l_@@_columns_width_dim
7373       {
7374         \use:c
7375         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7376       }
7377     }
7378   }
7379 }
```

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).

```
7380 {
7381   \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.

```
7382   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7383   {
7384     \bool_if:NT \l_@@_block_auto_columns_width_bool
7385     {
7386       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7387       \iow_shipout:Ne \@mainaux
7388       {
7389         \cs_gset:cpn
7390         { @@ _ 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.

```
7391         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7392       }
7393       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7394     }
7395   }
7396   \ignorespacesafterend
7397 }
```

## 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).

```

7398 \cs_new_protected:Npn \@@_create_extra_nodes:
7399 {
7400   \bool_if:nTF \l_@@_medium_nodes_bool
7401   {
7402     \bool_if:NTF \l_@@_no_cell_nodes_bool
7403     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7404     {
7405       \bool_if:NTF \l_@@_large_nodes_bool
7406       \@@_create_medium_and_large_nodes:
7407       \@@_create_medium_nodes:
7408     }
7409   }
7410   {
7411     \bool_if:NT \l_@@_large_nodes_bool
7412     {
7413       \bool_if:NTF \l_@@_no_cell_nodes_bool
7414       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7415       \@@_create_large_nodes:
7416     }
7417   }
7418 }

```

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`.

```

7419 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7420 {
7421   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7422   {
7423     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7424     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7425     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7426     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7427   }
7428   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7429   {
7430     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7431     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7432     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7433     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7434   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7435 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7436 {
7437     \int_step_variable:nnNn
7438     \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.

```

7439 {
7440     \cs_if_exist:cT
7441     { 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`.

```

7442 {
7443     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7444     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7445     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7446     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7447     {
7448         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7449         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7450     }

```

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`.

```

7451     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7452     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7453     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7454     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7455     {
7456         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7457         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7458     }
7459 }
7460 }
7461 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7462 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7463 {
7464     \dim_compare:nnNt
7465     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7466     {
7467         \@@_qpoint:n { row - \@@_i: - base }
7468         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7469         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7470     }
7471 }
7472 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7473 {
7474     \dim_compare:nnNt
7475     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7476     {
7477         \@@_qpoint:n { col - \@@_j: }
7478         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7479         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7480     }
7481 }
7482 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7483 \cs_new_protected:Npn \@@_create_medium_nodes:
7484 {
7485   \pgfpicture
7486   \pgfrememberpicturepositiononpagetrue
7487   \pgf@relevantforpicturesizefalse
7488   \@@_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”.

```

7489   \tl_set:Nn \l_@@_suffix_tl { -medium }
7490   \@@_create_nodes:
7491   \endpgfpicture
7492 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>15</sup>. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7493 \cs_new_protected:Npn \@@_create_large_nodes:
7494 {
7495   \pgfpicture
7496   \pgfrememberpicturepositiononpagetrue
7497   \pgf@relevantforpicturesizefalse
7498   \@@_computations_for_medium_nodes:
7499   \@@_computations_for_large_nodes:
7500   \tl_set:Nn \l_@@_suffix_tl { - large }
7501   \@@_create_nodes:
7502   \endpgfpicture
7503 }
7504 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7505 {
7506   \pgfpicture
7507   \pgfrememberpicturepositiononpagetrue
7508   \pgf@relevantforpicturesizefalse
7509   \@@_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”.

```

7510   \tl_set:Nn \l_@@_suffix_tl { - medium }
7511   \@@_create_nodes:
7512   \@@_computations_for_large_nodes:
7513   \tl_set:Nn \l_@@_suffix_tl { - large }
7514   \@@_create_nodes:
7515   \endpgfpicture
7516 }

```

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.

```

7517 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7518 {
7519   \int_set:Nn \l_@@_first_row_int 1
7520   \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`.

```

7521   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7522   {
7523     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7524     {
7525     (
7526         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7527         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7528     )
7529     / 2
7530     }
7531     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7532     { l_@@_row _ \@@_i: _ min_dim }
7533 }
7534 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7535 {
7536     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7537     {
7538     (
7539         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7540         \dim_use:c
7541         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7542     )
7543     / 2
7544     }
7545     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7546     { l_@@_column _ \@@_j: _ max _ dim }
7547 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7548     \dim_sub:cn
7549     { l_@@_column _ 1 _ min _ dim }
7550     \l_@@_left_margin_dim
7551     \dim_add:cn
7552     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7553     \l_@@_right_margin_dim
7554 }

```

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).

```

7555 \cs_new_protected:Npn \@@_create_nodes:
7556 {
7557     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7558     {
7559         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7560         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7561         \@@_pgf_rect_node:nnnnn
7562         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7563         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7564         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7565         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7566         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7567         \str_if_empty:NF \l_@@_name_str
7568         {
7569             \pgfnodealias
7570             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7571             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7572         }
7573     }
7574 }
7575 \int_step_inline:nn \c@iRow

```

```

7576 {
7577   \pgfnodealias
7578   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7579   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7580 }
7581 \int_step_inline:nn \c@jCol
7582 {
7583   \pgfnodealias
7584   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7585   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7586 }
7587 \pgfnodealias
7588 { \@@_env: - last - last \l_@@_suffix_tl }
7589 { \@@_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  $n$ .

```

7590   \seq_map_pairwise_function:NNN
7591   \g_@@_multicolumn_cells_seq
7592   \g_@@_multicolumn_sizes_seq
7593   \@@_node_for_multicolumn:nn
7594 }

7595 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7596 {
7597   \cs_set_nopar:Npn \@@_i: { #1 }
7598   \cs_set_nopar:Npn \@@_j: { #2 }
7599 }

```

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”).

```

7600 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7601 {
7602   \@@_extract_coords_values: #1 \q_stop
7603   \@@_pgf_rect_node:nnnnn
7604   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7605   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7606   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7607   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7608   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7609   \str_if_empty:NF \l_@@_name_str
7610   {
7611     \pgfnodealias
7612     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7613     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7614   }
7615 }

```

## 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).

```

7616 \keys_define:nn { nicematrix / Block / FirstPass }
7617 {
7618   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7619     \bool_set_true:N \l_@@_p_block_bool ,
7620   j .value_forbidden:n = true ,
7621   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7622   l .value_forbidden:n = true ,
7623   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7624   r .value_forbidden:n = true ,
7625   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7626   c .value_forbidden:n = true ,
7627   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7628   L .value_forbidden:n = true ,
7629   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7630   R .value_forbidden:n = true ,
7631   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7632   C .value_forbidden:n = true ,
7633   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7634   t .value_forbidden:n = true ,
7635   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7636   T .value_forbidden:n = true ,
7637   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7638   b .value_forbidden:n = true ,
7639   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7640   B .value_forbidden:n = true ,
7641   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7642   m .value_forbidden:n = true ,
7643   v-center .meta:n = m ,
7644   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7645   p .value_forbidden:n = true ,
7646   respect-arraystretch .code:n =
7647     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7648   respect-arraystretch .value_forbidden:n = true ,
7649   color .code:n =
7650     \@@_color:n { #1 }
7651     \tl_set_rescan:Nnn
7652       \l_@@_draw_tl
7653       { \char_set_catcode_other:N ! }
7654       { #1 } ,
7655   color .value_required:n = true ,
7656 }

```

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.

```

7657 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7658 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7659 {

```

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).

```

7660   \tl_if_blank:nTF { #2 }
7661   { \@@_Block_ii:nnnnn 1 1 }
7662   {
7663     \tl_if_in:nnTF { #2 } { - }
7664     {
7665       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7666       \@@_Block_i_czech:w \@@_Block_i:w
7667       #2 \q_stop
7668     }
7669     {
7670       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7671         \@@_Block_ii:nnnnn 1 1
7672     }
7673 }
7674 { #1 } { #3 } { #4 }
7675 \ignorespaces
7676 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7677 \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`.

```

7678 {
7679   \char_set_catcode_active:N -
7680   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7681 }

```

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.

```

7682 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7683 {

```

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).

```

7684   \bool_lazy_or:nnTF
7685     { \tl_if_blank_p:n { #1 } }
7686     { \str_if_eq_p:ee { * } { #1 } }
7687     { \int_set:Nn \l_tmpa_int { 100 } }
7688     { \int_set:Nn \l_tmpa_int { #1 } }
7689   \bool_lazy_or:nnTF
7690     { \tl_if_blank_p:n { #2 } }
7691     { \str_if_eq_p:ee { * } { #2 } }
7692     { \int_set:Nn \l_tmpb_int { 100 } }
7693     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7694   \int_compare:nNnTF \l_tmpb_int = 1
7695   {
7696     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7697       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7698       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7699   }
7700   { \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.

```

7701   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7702   \tl_set:Ne \l_tmpa_tl
7703   {
7704     { \int_use:N \c@iRow }
7705     { \int_use:N \c@jCol }
7706     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7707     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7708   }

```

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).

```
7709 \bool_set_false:N \l_tmpa_bool
7710 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
7711 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7712 \bool_case:nF
7713 {
7714 \l_tmpa_bool { \@@_Block_vii:eenenn }
7715 \l_@@_p_block_bool { \@@_Block_vi:eenenn }
```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```
7716 \l_@@_X_bool { \@@_Block_v:eenenn }
7717 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eenenn }
7718 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eenenn }
7719 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eenenn }
7720 }
7721 { \@@_Block_v:eenenn }
7722 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7723 }
```

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.

```
7724 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7725 {
7726 \int_gincr:N \g_@@_block_box_int
7727 \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7728 \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7729 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7730 {
7731 \tl_gput_right:Ne \g_@@_rules_tl
7732 {
7733 \@@_draw_diagbox:nnnnnn
7734 { \int_use:N \c@iRow }
7735 { \int_use:N \c@jCol }
7736 { \int_eval:n { \c@iRow + #1 - 1 } }
7737 { \int_eval:n { \c@jCol + #2 - 1 } }
7738 { \g_@@_row_style_tl \exp_not:n { ##1 } }
7739 { \g_@@_row_style_tl \exp_not:n { ##2 } }
7740 }
7741 }
7742 \box_gclear_new:c
7743 { 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!

```

7744   \hbox_gset:cn
7745   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7746   {

```

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`).

```

7747   \tl_if_empty:NTF \l_@@_color_tl
7748   { \int_compare:nNnT { #2 } = 1 \set@color }
7749   { \@@_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`.

```

7750   \int_compare:nNnT { #1 } = 1
7751   {
7752     \int_if_zero:nTF \c@iRow
7753     {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7754   \cs_set_eq:NN \Block \@@_NullBlock:
7755   \l_@@_code_for_first_row_tl
7756   }
7757   {
7758     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7759     {
7760       \cs_set_eq:NN \Block \@@_NullBlock:
7761       \l_@@_code_for_last_row_tl
7762     }
7763   }
7764   \g_@@_row_style_tl
7765   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7766   \@@_reset_arraystretch:
7767   \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7768   #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.).

```
7769 \@@_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}`.

```
7770 \bool_if:NTF \l_@@_tabular_bool
7771 {
7772   \bool_lazy_all:nTF
7773   {
7774     { \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.

```
7775     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7776     { ! \g_@@_rotate_bool }
7777   }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7778   {
7779     \use:e
7780     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7781       \exp_not:N \begin { minipage }
7782       [ \str_lowercase:f \l_@@_vpos_block_str ]
7783       { \l_@@_col_width_dim }
7784       \str_case:on \l_@@_hpos_block_str
7785       { c \centering r \raggedleft l \raggedright }
7786     }
7787     #5
7788   \end { minipage }
7789 }
```

In the other cases, we use a `{tabular}`.

```
7790   {
7791     \use:e
7792     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7793       \exp_not:N \begin { tabular }
7794       [ \str_lowercase:f \l_@@_vpos_block_str ]
7795       { @ { } \l_@@_hpos_block_str @ { } }
7796     }
7797     #5
7798   \end { tabular }
7799 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7801   {
7802     $ % $
7803     \use:e
7804     {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7805       \exp_not:N \begin { array }
7806       [ \str_lowercase:f \l_@@_vpos_block_str ]
7807       { @ { } \l_@@_hpos_block_str @ { } }
7808     }
7809     #5
```

```

7810         \end { array }
7811     $ % $
7812 }
7813 }

```

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).

```

7814 \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.

```

7815 \int_compare:nNnT { #2 } = 1
7816 {
7817     \dim_gset:Nn \g_@@_blocks_wd_dim
7818     {
7819         \dim_max:nn
7820         \g_@@_blocks_wd_dim
7821         {
7822             \box_wd:c
7823             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7824         }
7825     }
7826 }

```

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.

```

7827 \int_compare:nNnT { #1 } = 1
7828 {
7829     \bool_lazy_any:nT
7830     {
7831         { \str_if_empty_p:N \l_@@_vpos_block_str }
7832         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7833         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7834     }
7835     \@@_adjust_blocks_ht_dp:
7836 }
7837 \seq_gput_right:Ne \g_@@_blocks_seq
7838 {
7839     \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.

```

7840 {
7841     \exp_not:n { #3 } ,
7842     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7843 \bool_if:NT \g_@@_rotate_bool
7844 {
7845     \bool_if:NTF \g_@@_rotate_c_bool
7846     { m }
7847     {
7848         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7849         { T }
7850     }
7851 }
7852 }
7853 {
7854     \box_use_drop:c

```

```

7855         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7856     }
7857 }
7858 \bool_set_false:N \g_@@_rotate_c_bool
7859 }
7860 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7861 {
7862     \dim_gset:Nn \g_@@_blocks_ht_dim
7863     {
7864         \dim_max:nn
7865         \g_@@_blocks_ht_dim
7866         {
7867             \box_ht:c
7868             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7869         }
7870     }
7871     \dim_gset:Nn \g_@@_blocks_dp_dim
7872     {
7873         \dim_max:nn
7874         \g_@@_blocks_dp_dim
7875         {
7876             \box_dp:c
7877             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7878         }
7879     }
7880 }

7881 \cs_new:Npn \@@_adjust_hpos_rotate:
7882 {
7883     \bool_if:NT \g_@@_rotate_bool
7884     {
7885         \str_set:Ne \l_@@_hpos_block_str
7886         {
7887             \bool_if:NTF \g_@@_rotate_c_bool
7888             c
7889             {
7890                 \str_case:onF \l_@@_vpos_block_str
7891                 { b l B l t r T r }
7892                 {
7893                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7894                     r
7895                     l
7896                 }
7897             }
7898         }
7899     }
7900 }
7901 \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*.

```

7902 \cs_new_protected:Npn \@@_rotate_box_of_block:
7903 {
7904     \box_grotate:cn
7905     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7906     { 90 }
7907     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7908     {
7909         \vbox_gset_top:cn
7910         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7911         {

```

```

7912         \skip_vertical:n { 0.8 ex }
7913         \box_use:c
7914         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7915     }
7916 }
7917 \bool_if:NT \g_@@_rotate_c_bool
7918 {
7919     \hbox_gset:cn
7920     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7921     {
7922         $ % $
7923         \vcenter
7924         {
7925             \box_use:c
7926             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7927         }
7928         $ % $
7929     }
7930 }
7931 }

```

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.

```

7932 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7933 {
7934     \seq_gput_right:Ne \g_@@_blocks_seq
7935     {
7936         \l_tmpa_tl
7937         { \exp_not:n { #3 } }
7938         {
7939             \bool_if:NTF \l_@@_tabular_bool
7940             {
7941

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7942         \@@_reset_arraystretch:
7943         \exp_not:n
7944         {
7945             \dim_zero:N \extrarowheight
7946             #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).

```

7947         \tag_if_active:T { \tag_stop:n { table } }
7948         \use:e
7949         {
7950             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7951             { @ { } \l_@@_hpos_block_str @ { } }
7952         }
7953         #5
7954         \end { tabular }
7955     }
7956 }
7957 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7958         {
7959         {
The following will be no-op when respect-arraystretch is in force.
7960             \@@_reset_arraystretch:
7961             \exp_not:n
7962             {
7963                 \dim_zero:N \extrarowheight
7964                 #4
7965                 $ % $
7966                 \use:e
7967                 {
7968                     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7969                     { @ { } \l_@@_hpos_block_str @ { } }
7970                 }
7971                 #5
7972                 \end { array }
7973                 $ % $
7974             }
7975         }
7976     }
7977 }
7978 }
7979 }
7980 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7981 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7982 {
7983     \seq_gput_right:Ne \g_@@_blocks_seq
7984     {
7985         \l_tmpa_tl
7986         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7987         { { \exp_not:n { #4 #5 } } }
7988     }
7989 }
7990 \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`.

```

7991 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7992 {
7993     \seq_gput_right:Ne \g_@@_blocks_seq
7994     {
7995         \l_tmpa_tl
7996         { \exp_not:n { #3 } }
7997         { \exp_not:n { #4 #5 } }
7998     }
7999 }
8000 \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).

```

8001 \keys_define:nn { nicematrix / Block / SecondPass }
8002 {
8003     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8004     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8005     tikz .code:n =
8006         \IfPackageLoadedTF { tikz }
8007         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8008         { \@@_error:n { tikz~key~without~tikz } } ,
8009     tikz .value_required:n = true ,
8010     fill .code:n =
8011         \tl_set_rescan:Nnn
8012         \l_@@_fill_tl
8013         { \char_set_catcode_other:N ! }
8014         { #1 } ,
8015     fill .value_required:n = true ,

```

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

```

8016     opacity .tl_set:N = \l_@@_opacity_tl ,
8017     opacity .value_required:n = true ,
8018     draw .code:n =
8019         \tl_set_rescan:Nnn
8020         \l_@@_draw_tl
8021         { \char_set_catcode_other:N ! }
8022         { #1 } ,
8023     draw .default:n = default ,
8024     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8025     rounded-corners .default:n = 4 pt ,
8026     color .code:n =
8027         \@@_color:n { #1 }
8028         \tl_set_rescan:Nnn
8029         \l_@@_draw_tl
8030         { \char_set_catcode_other:N ! }
8031         { #1 } ,
8032     borders .clist_set:N = \l_@@_borders_clist ,
8033     borders .value_required:n = true ,
8034     hvlines .meta:n = { vlines , hlines } ,
8035     vlines .bool_set:N = \l_@@_vlines_block_bool ,
8036     hlines .bool_set:N = \l_@@_hlines_block_bool ,
8037     rules/width .dim_set:N = \arrayrulewidth ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8038     line-width .dim_set:N = \arrayrulewidth ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

8039     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8040         \bool_set_true:N \l_@@_p_block_bool ,
8041     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8042     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8043     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8044     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8045         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8046     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8047         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8048     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8049         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8050     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8051     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8052     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8053     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8054     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8055     m .value_forbidden:n = true ,
8056     v-center .meta:n = m ,
8057     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8058     p .value_forbidden:n = true ,
8059     name .str_set:N = \l_@@_block_name_str ,
8060     name .value_required:n = true ,
8061     respect-arraystretch .code:n =
8062         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,

```

```

8063     respect-arraystretch .value_forbidden:n = true ,
8064     transparent .bool_set:N = \l_@@_transparent_bool ,
8065     unknown .code:n =
8066         \@@_unknown_key:nn
8067         { nicematrix / Block / SecondPass }
8068         { Unknown-key~for~Block }
8069 }

```

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.

```

8070 \cs_new_protected:Npn \@@_draw_blocks:
8071 {
8072     \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8073     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8074 }
8075 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8076 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8077     \int_zero:N \l_@@_last_row_int
8078     \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”).

```

8079     \int_compare:nNnTF { #3 } > { 98 }
8080     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8081     { \int_set:Nn \l_@@_last_row_int { #3 } }
8082     \int_compare:nNnTF { #4 } > { 98 }
8083     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8084     { \int_set:Nn \l_@@_last_col_int { #4 } }
8085     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8086     {
8087         \bool_lazy_and:nnTF
8088         \l_@@_preamble_bool
8089         {
8090             \int_compare_p:n
8091             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8092         }
8093         {
8094             \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8095             \@@_msg_redirect_name:nn { Block-too~large~2 } { none }
8096             \@@_msg_redirect_name:nn { columns-not-used } { none }
8097         }
8098         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8099     }
8100     {
8101         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8102         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8103         {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8104         \use:e
8105         {
8106             \@@_Block_v:nnnnnn
8107             { #1 }
8108             { #2 }

```

```

8109         { \int_use:N \l_@@_last_row_int }
8110         { \int_use:N \l_@@_last_col_int }
8111     }
8112     { #5 }
8113     { #6 }
8114 }
8115 }
8116 }

```

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.

```

8117 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8118 {

```

The group is for the keys.

```

8119     \group_begin:
8120     \int_compare:nNnT { #1 } = { #3 }
8121     { \str_set:Nn \l_@@_vpos_block_str { t } }
8122     \keys_set:nn { nicematrix / Block / SecondPass } { #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`.

```

8123     \bool_if:NT \l_@@_amp_in_blocks_bool
8124     { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }
8125     \bool_lazy_and:nnT
8126     \l_@@_vlines_block_bool
8127     { ! \l_@@_ampersand_bool }
8128     {
8129         \tl_gput_right:Ne \g_@@_rules_tl
8130         {
8131             \@@_vlines_block:nnnnn
8132             { \dim_use:N \arrayrulewidth }
8133             { #1 } { #2 } { #3 } { #4 }
8134         }
8135     }
8136     \bool_if:NT \l_@@_hlines_block_bool
8137     {
8138         \tl_gput_right:Ne \g_@@_rules_tl
8139         {
8140             \@@_hlines_block:nnnnn
8141             { \dim_use:N \arrayrulewidth }
8142             { #1 } { #2 } { #3 } { #4 }
8143         }
8144     }

```

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).

```

8145     \bool_if:NF \l_@@_transparent_bool
8146     {
8147         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8148         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8149     }

8150     \tl_if_empty:NF \l_@@_draw_tl
8151     {
8152         \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
8153         { \@@_error:n { hlines~with~color } }
8154         \tl_gput_right:Nn \g_@@_rules_tl
8155         {
8156             \@@_stroke_block:nnnnn

```

#5 are the options

```

8157         { #5 } { #1 } { #2 } { #3 } { #4 }
8158     }
8159     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8160     { { #1 } { #2 } { #3 } { #4 } }
8161 }
8162 \clist_if_empty:NF \l_@@_borders_clist
8163 {
8164     \tl_gput_right:Nn \g_nicematrix_code_after_tl
8165     {
8166         \@@_stroke_borders_block:nnnnn
8167         { #5 } { #1 } { #2 } { #3 } { #4 }
8168     }
8169 }
8170 \tl_if_empty:NF \l_@@_fill_tl
8171 {
8172     \@@_add_opacity_to_fill:
8173     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8174     {
8175         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8176         { #1 - #2 }
8177         { #3 - #4 }
8178         { \dim_use:N \l_@@_rounded_corners_dim }
8179     }
8180 }
8181 \seq_if_empty:NF \l_@@_tikz_seq
8182 {
8183     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8184     {
8185         \@@_block_tikz:nnnnn
8186         { \seq_use:Nn \l_@@_tikz_seq { , } }
8187         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8188     }
8189 }
8190 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8191 {
8192     \tl_gput_right:Ne \g_@@_rules_tl
8193     {
8194         \@@_draw_diagbox:nnnnnn
8195         { #1 } { #2 } { #3 } { #4 }
8196         { \exp_not:n { ##1 } }
8197         { \exp_not:n { ##2 } }
8198     }
8199 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & four & five & \\\
six & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
		two
three	four	five
six	seven	eight

We highlight the node 1-1-block-short

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

8200 \pgfpicture
8201 \pgfrememberpicturepositiononpagetrue
8202 \pgf@relevantforpicturesizefalse
8203 \@@_qpoint:n { row - #1 }
8204 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8205 \@@_qpoint:n { col - #2 }
8206 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8207 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8208 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8209 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8210 \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.

```

8211 \@@_pgf_rect_node:nnnnn
8212 { \@@_env: - #1 - #2 - block }
8213 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8214 \str_if_empty:NF \l_@@_block_name_str
8215 {
8216 \pgfnodealias
8217 { \@@_env: - \l_@@_block_name_str }
8218 { \@@_env: - #1 - #2 - block }
8219 \str_if_empty:NF \l_@@_name_str
8220 {
8221 \pgfnodealias
8222 { \l_@@_name_str - \l_@@_block_name_str }
8223 { \@@_env: - #1 - #2 - block }
8224 }
8225 }

```

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.

```

8226 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8227 {
8228 \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.

```

8229 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8230 {

```

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.

```

8231 \cs_if_exist:cT
8232 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8233 {
8234 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8235 {
8236 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8237 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8238 }
8239 }
8240 }

```

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.

```

8241     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8242     {
8243         \@@_qpoint:n { col - #2 }
8244         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8245     }
8246     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8247     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8248     {
8249         \cs_if_exist:cT
8250         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8251         {
8252             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8253             {
8254                 \pgfpointanchor
8255                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8256                 { east }
8257                 \dim_set:Nn \l_@@_tmpd_dim
8258                 { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8259             }
8260         }
8261     }
8262     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8263     {
8264         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8265         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8266     }
8267     \@@_pgf_rect_node:nnnnn
8268     { \@@_env: - #1 - #2 - block - short }
8269     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8270 }

```

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.

```

8271     \bool_if:NT \l_@@_medium_nodes_bool
8272     {
8273         \@@_pgf_rect_node:nnn
8274         { \@@_env: - #1 - #2 - block - medium }
8275         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8276         {
8277             \pgfpointanchor
8278             { \@@_env:
8279               - \int_use:N \l_@@_last_row_int
8280               - \int_use:N \l_@@_last_col_int - medium
8281             }
8282             { south-east }
8283         }
8284     }
8285     \endpgfpicture
8286

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand `&`.

```

8287     \bool_if:NTF \l_@@_ampersand_bool
8288     {
8289         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8290         \int_zero_new:N \l_@@_split_int
8291         \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).

```

8292     \int_set:Nn \l_@@_first_row_int { #1 }

```

```

8293 \int_set:Nn \l_@@_first_col_int { #2 }
8294 \int_set:Nn \l_@@_last_row_int { #3 }
8295 \int_set:Nn \l_@@_last_col_int { #4 }

8296 \pgfpicture
8297 \pgfrememberpicturepositiononpagetrue
8298 \pgf@relevantforpicturesizefalse
8299 \@@_qpoint:n { row - #1 }
8300 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8301 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8302 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8303 \@@_qpoint:n { col - #2 }
8304 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8305 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8306 \dim_set:Nn \l_tmpb_dim
8307 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8308 \bool_lazy_or:nnT
8309 \l_@@_vlines_block_bool
8310 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8311 {
8312 \int_step_inline:nn { \l_@@_split_int - 1 }
8313 {
8314 \pgfpathmoveto
8315 {
8316 \pgfpoint
8317 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8318 \l_@@_tmpc_dim
8319 }
8320 \pgfpathlineto
8321 {
8322 \pgfpoint
8323 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8324 \l_@@_tmpd_dim
8325 }
8326 \CT@arc@
8327 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8328 \pgfsetrectcap
8329 \pgfusepathqstroke
8330 }
8331 }
8332 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8333 \int_zero_new:N \l_@@_split_i_int

8334 \str_if_eq:eeTF \l_@@_vpos_block_str T
8335 {
8336 \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8337 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8338 }
8339 {
8340 \str_if_eq:eeTF \l_@@_vpos_block_str B
8341 {
8342 \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8343 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8344 }
8345 {
8346 \bool_lazy_or:nnTF
8347 { \int_compare_p:nNn { #1 } = { #3 } }
8348 { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8349 {
8350 \@@_qpoint:n { row - #1 - base }
8351 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8352 }
8353 {
8354 \str_if_eq:eeTF \l_@@_vpos_block_str b
8355 {

```

```

8356         \@@_qpoint:n { row - #3 - base }
8357         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8358     }
8359 {
8360     \@@_qpoint:n { #1 - #2 - block }
8361     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8362 }
8363 }
8364 }
8365 }
8366 \int_step_inline:nn \l_@@_split_int
8367 {
8368     \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8369     \int_set:Nn \l_@@_split_i_int { ##1 }
8370     \dim_set:Nn \col@sep
8371     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8372     \pgftransformshift
8373     {
8374         \pgfpoint
8375         {
8376             \l_tmpa_dim + ##1 \l_tmpb_dim -
8377             \str_case:on \l_@@_hpos_block_str
8378             {
8379                 l { \l_tmpb_dim + \col@sep }
8380                 c { 0.5 \l_tmpb_dim }
8381                 r { \col@sep }
8382             }
8383         }
8384         { \l_@@_tmpc_dim }
8385     }
8386     \pgfset { inner~sep = \c_zero_dim }
8387     \pgfnode
8388     { rectangle }
8389     {
8390         \str_if_eq:eeTF T \l_@@_vpos_block_str
8391         {
8392             \str_case:on \l_@@_hpos_block_str
8393             {
8394                 l { north-west }
8395                 c { north }
8396                 r { north-east }
8397             }
8398         }
8399         {
8400             \str_if_eq:eeTF B \l_@@_vpos_block_str
8401             {
8402                 \str_case:on \l_@@_hpos_block_str
8403                 {
8404                     l { south-west }
8405                     c { south }
8406                     r { south-east }
8407                 }
8408             }
8409         }
8410         \bool_lazy_any:nTF
8411         {
8412             { \int_compare_p:nNn { #1 } = { #3 } }
8413             { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8414             { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8415         }
8416         {
8417             \str_case:on \l_@@_hpos_block_str

```

```

8418         {
8419             l { base~west }
8420             c { base }
8421             r { base~east }
8422         }
8423     }
8424     {
8425         \str_case:on \l_@@_hpos_block_str
8426         {
8427             l { west }
8428             c { center }
8429             r { east }
8430         }
8431     }
8432 }
8433 }
8434 }
8435 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8436 \group_end:
8437 }
8438 \endpgfpicture
8439 }

```

Now the case where there is no ampersand & in the content of the block.

```

8440 {
8441     \bool_if:NTF \l_@@_p_block_bool
8442     {

```

When the final user has used the key p, we have to compute the width.

```

8443     \pgfpicture
8444     \pgfrememberpicturepositiononpagetrue
8445     \pgf@relevantforpicturesizefalse
8446     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8447     {
8448         \@@_qpoint:n { col - #2 }
8449         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8450         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8451     }
8452     {
8453         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8454         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8455         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8456     }
8457     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8458     \endpgfpicture
8459     \hbox_set:Nn \l_@@_cell_box
8460     {
8461         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8462             { \g_tmpb_dim }
8463             \str_case:on \l_@@_hpos_block_str
8464             { c \centering r \raggedleft l \raggedright j { } }
8465             #6
8466             \end { minipage }
8467         }
8468     }
8469     { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8470     \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.

```

8471     \pgfpicture
8472     \pgfrememberpicturepositiononpagetrue
8473     \pgf@relevantforpicturesizefalse
8474     \bool_lazy_any:nTF

```

```

8475     {
8476     { \str_if_empty_p:N \l_@@_vpos_block_str }
8477     { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8478     { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8479     { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8480     }

```

```

8481     {

```

If we are in the “first column”, we must put the block as if it was with the key r.

```

8482     \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.

```

8483     \bool_if:nT \g_@@_last_col_found_bool
8484     {
8485     \int_compare:nNnT { #2 } = \g_@@_col_total_int
8486     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8487     }

```

\l\_tmpa\_tl will contain the anchor of the PGF node which will be used.

```

8488     \tl_set:Ne \l_tmpa_tl
8489     {
8490     \str_case:on \l_@@_vpos_block_str
8491     {

```

We recall that \l\_@@\_vpos\_block\_str is empty when the user has not used a key for the vertical position of the block.

```

8492     { } {
8493     \str_case:on \l_@@_hpos_block_str
8494     {
8495     c { center }
8496     l { west }
8497     r { east }
8498     j { center }
8499     }
8500     }
8501     c {
8502     \str_case:on \l_@@_hpos_block_str
8503     {
8504     c { center }
8505     l { west }
8506     r { east }
8507     j { center }
8508     }
8509     }
8510     }
8511     T {
8512     \str_case:on \l_@@_hpos_block_str
8513     {
8514     c { north }
8515     l { north-west }
8516     r { north-east }
8517     j { north }
8518     }
8519     }
8520     }
8521     B {
8522     \str_case:on \l_@@_hpos_block_str
8523     {
8524     c { south }
8525     l { south-west }
8526     r { south-east }
8527     j { south }
8528     }
8529     }

```

```

8530         }
8531     }
8532 }
8533 \pgftransformshift
8534 {
8535     \pgfpointanchor
8536     {
8537         \@@_env: - #1 - #2 - block
8538         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8539     }
8540     \l_tmpa_tl
8541 }
8542 \pgfset { inner~sep = \c_zero_dim }
8543 \pgfnode
8544 { rectangle }
8545 \l_tmpa_tl
8546 { \box_use_drop:N \l_@@_cell_box } { } { }
8547 }

```

End of the case when  $\l_@@\_vpos\_block\_str$  is equal to c, T or B. Now, the other cases.

```

8548 {
8549     \pgfextracty \l_tmpa_dim
8550     {
8551         \@@_qpoint:n
8552         {
8553             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8554             - base
8555         }
8556     }
8557     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in  $\pgf@x$ ) the  $x$ -value of the center of the block.

```

8558 \pgfpointanchor
8559 {
8560     \@@_env: - #1 - #2 - block
8561     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8562 }
8563 {
8564     \str_case:on \l_@@_hpos_block_str
8565     {
8566         c { center }
8567         l { west }
8568         r { east }
8569         j { center }
8570     }
8571 }

```

We put the label of the block which has been composed in  $\l_@@\_cell\_box$ .

```

8572 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8573 \pgfset { inner~sep = \c_zero_dim }
8574 \pgfnode
8575 { rectangle }
8576 {
8577     \str_case:on \l_@@_hpos_block_str
8578     {
8579         c { base }
8580         l { base~west }
8581         r { base~east }
8582         j { base }
8583     }
8584 }
8585 { \box_use_drop:N \l_@@_cell_box } { } { }
8586 }

```

```

8587         \endpgfpicture
8588     }
8589 \group_end:
8590 }
8591 \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).

```

8592 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8593 {
8594     \tl_if_empty:NF \l_@@_opacity_tl
8595     {
8596         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8597             {
8598                 \tl_set:Nx \l_@@_fill_tl
8599                 {
8600                     [ opacity = \l_@@_opacity_tl ,
8601                     \tl_tail:o \l_@@_fill_tl
8602                 }
8603             }
8604             {
8605                 \tl_set:Nx \l_@@_fill_tl
8606                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8607             }
8608         }
8609     }

```

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*.

```

8610 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8611 {
8612     \group_begin:
8613     \tl_clear:N \l_@@_draw_tl
8614     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8615     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8616     \tl_if_empty:NF \l_@@_draw_tl
8617     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8618         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8619         \CT@arc@
8620         { \@@_color:o \l_@@_draw_tl }
8621     }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8622 \pgfsetcornersarced
8623 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8624 \int_compare:nNnF { #2 } > \c@iRow
8625 {
8626     \int_compare:nNnF { #3 } > \c@jCol
8627     {
8628         \@@_qpoint:n { row - #2 }
8629         \dim_set_eq:NN \l_tmpb_dim \pgf@y
8630         \@@_qpoint:n { col - #3 }
8631         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8632         \@@_qpoint:n
8633         { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8634         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8635         \@@_qpoint:n
8636         { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8637         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

```

8638         \pgfpathrectanglecorners
8639         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8640         { \pgfpoint \pgf@x \l_tmpa_dim }
8641         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8642         \pgfusepathqstroke
8643         { \pgfusepath { stroke } }
8644     }
8645 }
8646 \group_end:
8647 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8648 \keys_define:nn { nicematrix / BlockStroke }
8649 {
8650     color .tl_set:N = \l_@@_draw_tl ,
8651     draw .code:n =
8652         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8653     draw .default:n = default ,
8654     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8655     rounded-corners .default:n = 4 pt ,
8656     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8657     line-width .dim_set:N = \l_@@_line_width_dim
8658 }

```

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 other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8659 \cs_new_protected:Npn \@@_vlines_block:nnnnn #1 #2 #3 #4 #5
8660 {
8661     \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).

```

8662     \dim_set:Nn \arrayrulewidth { #1 }
8663     \pgfsetlinewidth \arrayrulewidth % added 2026-03-28

```

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).

```

8664     \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8665     {
8666         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8667         ||
8668         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8669         ||
8670         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8671         ||
8672         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8673     }
8674     \bool_if:NT \l_@@_fix_vertex_bool
8675     {
8676         \int_compare:nNnT { #3 } > 1
8677         {
8678             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8679             {
8680                 { 1 }
8681                 { 1 }
8682                 { \int_use:N \c@iRow }
8683                 { \int_eval:n { #3 -1 } }

```

```

8684         { }
8685     }
8686 }
8687 \int_compare:nNt { #5 } < \c@jCol
8688 {
8689     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8690     {
8691         { 1 }
8692         { \int_eval:n { #5 + 1 } }
8693         { \int_use:N \c@iRow }
8694         { \int_use:N \c@jCol }
8695     }
8696 }
8697 }
8698 }
8699 \int_set:Nn \l_@@_start_int { #2 }
8700 \int_set:Nn \l_@@_end_int { #4 }
8701 \int_step_inline:nnn { #3 } { #5 + 1 }
8702 {
8703     \int_set:Nn \l_@@_position_int { ##1 }
8704     \socket_use:n { nicematrix / draw-vrule }
8705 }
8706 \group_end:
8707 }

```

The command `\@@_hlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8708 \cs_new_protected:Npn \@@_hlines_block:nnnnn #1 #2 #3 #4 #5
8709 {
8710     \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).

```

8711     \dim_set:Nn \arrayrulewidth { #1 }
8712     \pgfsetlinewidth \arrayrulewidth % added 2026-03-28

```

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).

```

8713     \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8714     {
8715         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #2 }
8716         ||
8717         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #3 }
8718         ||
8719         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #4 }
8720         ||
8721         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #5 }
8722     }
8723     \bool_if:NT \l_@@_fix_vertex_bool
8724     {
8725         \int_compare:nNt { #2 } > 1
8726         {
8727             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8728             {
8729                 { 1 }
8730                 { 1 }
8731                 { \int_eval:n { #2 - 1 } }
8732                 { \int_use:N \c@jCol }
8733             }
8734         }

```

```

8735     }
8736     \int_compare:nNtT { #4 } < \c@iRow
8737     {
8738         \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8739         {
8740             { \int_eval:n { #4 + 1 } }
8741             { 1 }
8742             { \int_use:N \c@iRow }
8743             { \int_use:N \c@jCol }
8744             { }
8745         }
8746     }
8747 }
8748 \int_set:Nn \l_@@_start_int { #3 }
8749 \int_set:Nn \l_@@_end_int { #5 }
8750 \int_step_inline:nnn { #2 } { #4 + 1 }
8751 {
8752     \int_set:Nn \l_@@_position_int { ##1 }
8753     \socket_use:n { nicematrix / draw-hrule }
8754 }
8755 \group_end:
8756 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8757 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8758 {
8759     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8760     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8761     \dim_compare:nNtTF \l_@@_rounded_corners_dim > \c_zero_dim
8762     { \@@_error:n { borders~forbidden } }
8763     {
8764         \tl_clear_new:N \l_@@_borders_tikz_tl
8765         \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8766         \tl_set:Nn \l_@@_tmpc_tl { #2 }
8767         \tl_set:Nn \l_@@_tmpd_tl { #3 }
8768         \tl_set:Nn \l_tmpa_tl { \int_eval:n { #4 + 1 } }
8769         \tl_set:Nn \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8770         \@@_stroke_borders_block_i:
8771     }
8772 }
8773 \AtBeginDocument
8774 {
8775     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8776     {
8777         \c_@@_pgfortikzpicture_tl
8778         \@@_stroke_borders_block_ii:
8779         \c_@@_endpgfortikzpicture_tl
8780     }
8781 }
8782 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8783 {
8784     \pgfrememberpicturepositiononpagetrue
8785     \pgf@relevantforpicturesizefalse
8786     \CT@arc@
8787     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8788     \clist_if_in:NnT \l_@@_borders_clist { right }
8789     { \@@_stroke_vertical:n \l_tmpb_tl }
8790     \clist_if_in:NnT \l_@@_borders_clist { left }
8791     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8792     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8793     { \@@_stroke_horizontal:n \l_tmpa_tl }

```

```

8794 \clist_if_in:NnT \l_@@_borders_clist { top }
8795 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8796 }
8797 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8798 {
8799   tikz .code:n =
8800     \cs_if_exist:NTF \tikzpicture
8801     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8802     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8803   tikz .value_required:n = true ,
8804   top .code:n = ,
8805   bottom .code:n = ,
8806   left .code:n = ,
8807   right .code:n = ,
8808   unknown .code:n = \@@_error:n { bad~border }
8809 }

```

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).

```

8810 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8811 {
8812   \@@_qpoint:n \l_@@_tmpc_tl
8813   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8814   \@@_qpoint:n \l_tmpa_tl
8815   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8816   \@@_qpoint:n { #1 }
8817   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8818   {
8819     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8820     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8821     \pgfusepath{stroke}
8822   }
8823   {
8824     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8825     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8826   }
8827 }

```

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).

```

8828 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8829 {
8830   \@@_qpoint:n \l_@@_tmpd_tl
8831   \clist_if_in:NnTF \l_@@_borders_clist { left }
8832   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8833   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8834   \@@_qpoint:n \l_tmpb_tl
8835   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8836   \@@_qpoint:n { #1 }
8837   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8838   {
8839     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8840     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8841     \pgfusepath{stroke}
8842   }
8843   {
8844     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8845     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8846   }
8847 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

8848 \keys_define:nn { nicematrix / BlockBorders }
8849 {
8850   borders .clist_set:N = \l_@@_borders_clist ,
8851   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8852   rounded-corners .default:n = 4 pt ,
8853   rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

8854   line-width .dim_set:N = \l_@@_line_width_dim ,
8855 }

```

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.

```

8856 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8857 {
8858   \begin { tikzpicture }
8859   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8860   \clist_map_inline:nn { #1 }
8861   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8862     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8863     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8864     (
8865       [
8866         xshift = \dim_use:N \l_@@_offset_dim ,
8867         yshift = - \dim_use:N \l_@@_offset_dim
8868       ]
8869       #2 -| #3
8870     )
8871     rectangle
8872     (
8873       [
8874         xshift = - \dim_use:N \l_@@_offset_dim ,
8875         yshift = \dim_use:N \l_@@_offset_dim
8876       ]
8877       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8878     ) ;
8879   }
8880   \end { tikzpicture }
8881 }
8882 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8883 \keys_define:nn { nicematrix / SpecialOffset }
8884 {
8885   offset .dim_set:N = \l_@@_offset_dim ,
8886 }

```

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.

```

8887 \cs_new_protected:Npn \@@_NullBlock:
8888 { \@@_collect_options:n { \@@_NullBlock_i: } }
8889 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8890 { }

```

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.

```

8891 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8892 {
8893   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8894   {

```

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`).

```

8895     \@@_subcellcolor:nnnnnnn
8896     {
8897       \tl_if_blank:nTF { #1 }
8898       { { \exp_not:n { #2 } } }
8899       { [ #1 ] { \exp_not:n { #2 } } }
8900     }
8901     { \int_use:N \l_@@_first_row_int } % first row of the block
8902     { \int_use:N \l_@@_first_col_int } % first column of the block
8903     { \int_use:N \l_@@_last_row_int } % last row of the block
8904     { \int_use:N \l_@@_last_col_int } % last column of the block
8905     { \int_use:N \l_@@_split_int }
8906     { \int_use:N \l_@@_split_i_int }
8907   }
8908   \ignorespaces
8909 }

8910 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8911 {
8912   \@@_color_opacity: #1
8913   \pgfpicture
8914   \pgf@relevantforpicturesizefalse
8915   \@@_qpoint:n { col - #3 }
8916   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8917   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8918   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8919   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8920   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8921   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8922   \@@_qpoint:n { row - #2 }
8923   \pgfpathrectanglecorners
8924     { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
8925     { \pgfpoint \l_tmpb_dim \pgf@y }
8926   \pgfusepathqfill
8927   \endpgfpicture
8928 }

```

## 27 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8929 \keys_define:nn { nicematrix / Auto }
8930 {
8931   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8932   columns-type .value_required:n = true ,
8933   l .meta:n = { columns-type = l } ,
8934   r .meta:n = { columns-type = r } ,
8935   c .meta:n = { columns-type = c } ,
8936   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8937   delimiters / color .value_required:n = true ,
8938   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8939   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,

```

```

8940     delimiters .value_required:n = true ,
8941     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8942     rounded-corners .default:n = 4 pt
8943 }
8944 \NewDocumentCommand \AutoNiceMatrixWithDelims
8945 { m m 0 { } } > { \SplitArgument { 1 } { - } } m 0 { } m ! 0 { } }
8946 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8947 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8948 {

```

The group is for the protection of the keys.

```

8949     \group_begin:
8950     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8951     \use:e
8952     {
8953         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8954         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8955         [ \exp_not:o \l_tmpa_tl ]
8956     }
8957     \int_if_zero:nT \l_@@_first_row_int
8958     {
8959         \int_if_zero:nT \l_@@_first_col_int { & }
8960         \prg_replicate:nn { #4 - 1 } { & }
8961         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8962     }
8963     \prg_replicate:nn { #3 }
8964     {
8965         \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).

```

8966         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8967         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8968     }
8969     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8970     {
8971         \int_if_zero:nT \l_@@_first_col_int { & }
8972         \prg_replicate:nn { #4 - 1 } { & }
8973         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8974     }
8975     \end { NiceArrayWithDelims }
8976     \group_end:
8977 }
8978 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8979 {
8980     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8981     {
8982         \bool_gset_true:N \g_@@_delims_bool
8983         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8984         \AutoNiceMatrixWithDelims { #2 } { #3 }
8985     }
8986 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8987 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8988 {
8989     \group_begin:
8990     \bool_gset_false:N \g_@@_delims_bool
8991     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8992     \group_end:
8993 }

```

## 28 The redefinition of the command `\dotfill`

```

8994 \cs_set_eq:NN \@_old_dotfill: \dotfill
8995 \cs_new_protected:Npn \@_dotfill:
8996 {

```

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}`).

```

8997 \@_old_dotfill:
8998 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@_dotfill_i:
8999 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

9000 \cs_new_protected:Npn \@_dotfill_i:
9001 {
9002   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9003     { \@_old_dotfill: }
9004 }

```

## 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.

```

9005 \cs_new_protected:Npn \@_diagbox:nn #1 #2
9006 {
9007   \tl_gput_right:Ne \g_@@_rules_tl
9008   {
9009     \@_draw_diagbox:nnnnnn
9010     { \int_use:N \c@iRow }
9011     { \int_use:N \c@jCol }
9012     { \int_use:N \c@iRow }
9013     { \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.

```

9014     { \g_@@_row_style_tl \exp_not:n { #1 } }
9015     { \g_@@_row_style_tl \exp_not:n { #2 } }
9016   }

```

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`.

```

9017   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9018   {
9019     { \int_use:N \c@iRow }
9020     { \int_use:N \c@jCol }
9021     { \int_use:N \c@iRow }
9022     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9023     { }
9024   }
9025 }

```

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.

```

9026 \cs_new_protected:Npn \@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9027 {

```

```

9028 \group_begin:
9029 \@@_qpoint:n { row - #1 }
9030 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9031 \@@_qpoint:n { col - #2 }
9032 \dim_set_eq:NN \l_tmpb_dim \pgf@x
9033 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9034 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9035 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9036 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9037 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9038 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9039 {

```

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.

```

9040 \CT@arc@
9041 \pgfsetroundcap
9042 \pgfusepathqstroke
9043 }
9044 \pgfset { inner~sep = 1 pt }
9045 \pgfscope
9046 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9047 \pgfnode { rectangle } { south~west }
9048 {
9049 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

9050 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9051 \end { minipage }
9052 }
9053 { }
9054 { }
9055 \endpgfscope
9056 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9057 \pgfnode { rectangle } { north~east }
9058 {
9059 \begin { minipage } { 20 cm }
9060 \raggedleft
9061 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9062 \end { minipage }
9063 }
9064 { }
9065 { }
9066 \group_end:
9067 }

```

## 30 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 89.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

9068 \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 `\`.

```

9069 \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`.

```

9070 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9071 {
9072   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9073   \@@_CodeAfter_iv:n
9074 }

```

We catch the argument of the command `\end` (in `#1`).

```

9075 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9076 {

```

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.

```

9077   \str_if_eq:eeTF \@currenvir { #1 }
9078   { \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`.

```

9079   {
9080     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9081     \@@_CodeAfter_ii:n
9082   }
9083 }

```

## 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.

```

9084 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9085 {
9086   \pgfpicture
9087   \pgfrememberpicturepositiononpagetrue
9088   \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.

```

9089   \@@_qpoint:n { row - 1 }
9090   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9091   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9092   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

We will compute in `\l_tmpa_dim` the  $x$ -value where we will have to put our delimiter (on the left side or on the right side).

```

9093   \bool_if:nTF { #3 }
9094   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9095   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9096   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9097   {
9098     \cs_if_exist:cT
9099     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

9100     {
9101         \pgfpointanchor
9102         { \@@_env: - ##1 - #2 }
9103         { \bool_if:nTF { #3 } { west } { east } }
9104         \dim_set:Nn \l_tmpa_dim
9105         {
9106             \bool_if:nTF { #3 }
9107             \dim_min:nn
9108             \dim_max:nn
9109             \l_tmpa_dim
9110             \pgf@x
9111         }
9112     }
9113 }

```

Now we can put the delimiter with a node of PGF.

```

9114     \pgfset { inner~sep = \c_zero_dim }
9115     \dim_zero:N \nulldelimiterspace
9116     \pgftransformshift
9117     {
9118         \pgfpoint
9119         \l_tmpa_dim
9120         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9121     }
9122     \pgfnode
9123     { rectangle }
9124     { \bool_if:nTF { #3 } { east } { west } }
9125     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9126         \nullfont
9127         $ % $
9128         \@@_color:o \l_@@_delimiters_color_tl
9129         \bool_if:nTF { #3 } { \left #1 } { \left . }
9130         \vcenter
9131         {
9132             \nullfont
9133             \hrule \@height
9134                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9135                 \@depth \c_zero_dim
9136                 \@width \c_zero_dim
9137         }
9138         \bool_if:nTF { #3 } { \right . } { \right #1 }
9139         $ % $
9140     }
9141     { }
9142     { }
9143     \endpgfpicture
9144 }

```

## 32 The command `\SubMatrix`

```

9145 \keys_define:nn { nicematrix / sub-matrix }
9146 {
9147     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9148     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9149     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9150     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9151     xshift .value_required:n = true ,
9152     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9153     delimiters / color .value_required:n = true ,

```

```

9154     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9155     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9156     hlines .default:n = all ,
9157     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9158     vlines .default:n = all ,
9159     hvlines .meta:n = { hlines, vlines } ,
9160     hvlines .value_forbidden:n = true
9161   }
9162 \keys_define:nn { nicematrix }
9163 {
9164   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9165   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9166   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9167   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9168 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

9169 \keys_define:nn { nicematrix / SubMatrix }
9170 {
9171   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9172   delimiters / color .value_required:n = true ,
9173   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9174   hlines .default:n = all ,
9175   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9176   vlines .default:n = all ,
9177   hvlines .meta:n = { hlines, vlines } ,
9178   hvlines .value_forbidden:n = true ,
9179   name .code:n =
9180     \tl_if_empty:nTF { #1 }
9181     { \@@_error:n { Invalid-name } }
9182     {
9183       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9184       {
9185         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9186         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9187         {
9188           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9189           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9190         }
9191       }
9192       { \@@_error:n { Invalid-name } }
9193     } ,
9194   name .value_required:n = true ,
9195   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9196   rules .value_required:n = true ,
9197   code .tl_set:N = \l_@@_code_tl ,
9198   code .value_required:n = true ,
9199   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
9200 }

9201 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9202 {
9203   \tl_gput_right:Ne \g_@@_pre_code_after_tl
9204   {
9205     \SubMatrix { #1 } { #2 } { #3 } { #4 }
9206     [
9207       delimiters / color = \l_@@_delimiters_color_tl ,
9208       hlines = \l_@@_submatrix_hlines_clist ,
9209       vlines = \l_@@_submatrix_vlines_clist ,
9210       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9211       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9212       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,

```

```

9213         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9214         #5
9215     ]
9216 }
9217 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9218 \ignorespaces
9219 }

9220 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9221 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9222 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9223 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9224 {
9225     \seq_gput_right:Ne \g_@@_submatrix_seq
9226     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9227     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9228     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9229     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9230     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9231 }
9232 }

```

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).

```

9233 \NewDocumentCommand \@@_compute_i_j:nn
9234 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9235 { \@@_compute_i_j:nnnn #1 #2 }

9236 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9237 {
9238     \def \l_@@_first_i_tl { #1 }
9239     \def \l_@@_first_j_tl { #2 }
9240     \def \l_@@_last_i_tl { #3 }
9241     \def \l_@@_last_j_tl { #4 }
9242     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9243     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9244     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9245     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9246     \tl_if_eq:NnT \l_@@_last_i_tl { last }
9247     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9248     \tl_if_eq:NnT \l_@@_last_j_tl { last }
9249     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9250 }

```

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`.

```

9251 \AtBeginDocument
9252 {
9253   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
9254   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9255     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9256 }
9257 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
9258 {
9259   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9260 \@@_compute_i_j:nn { #2 } { #3 }
9261 \int_compare:nNt \l_@@_first_i_tl = \l_@@_last_i_tl
9262 { \def \arraystretch { 1 } }
9263 \bool_lazy_or:nnTF
9264 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9265 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9266 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9267 {
9268   \str_clear_new:N \l_@@_submatrix_name_str
9269   \keys_set:nn { nicematrix / SubMatrix } { #5 }
9270   \pgfpicture
9271   \pgfrememberpicturepositiononpagetrue
9272   \pgf@relevantforpicturesizefalse
9273   \pgfset { inner~sep = \c_zero_dim }
9274   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9275   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9276 \bool_if:NtF \l_@@_submatrix_slim_bool
9277 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9278 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9279 {
9280   \cs_if_exist:cT
9281   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9282   {
9283     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9284     \dim_compare:nNt \pgf@x < \l_@@_x_initial_dim
9285     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9286   }
9287   \cs_if_exist:cT
9288   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9289   {
9290     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9291     \dim_compare:nNt \pgf@x > \l_@@_x_final_dim
9292     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9293   }
9294 }
9295 \dim_compare:nNtF \l_@@_x_initial_dim = \c_max_dim
9296 { \@@_error:nn { Impossible~delimiter } { left } }
9297 {
9298   \dim_compare:nNtF \l_@@_x_final_dim = { - \c_max_dim }
9299   { \@@_error:nn { Impossible~delimiter } { right } }
9300   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9301 }
9302 \endpgfpicture
9303 }
9304 \group_end:
9305 \ignorespaces
9306 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9307 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9308 {
9309   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9310   \dim_set:Nn \l_@@_y_initial_dim
9311   {
9312     \fp_to_dim:n
9313     {
9314       \pgf@y
9315       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9316     }
9317   }
9318   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9319   \dim_set:Nn \l_@@_y_final_dim
9320   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9321   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9322   {
9323     \cs_if_exist:cT
9324     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9325     {
9326       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9327       \dim_set:Nn \l_@@_y_initial_dim
9328       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9329     }
9330     \cs_if_exist:cT
9331     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9332     {
9333       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9334       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9335       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9336     }
9337   }
9338   \dim_set:Nn \l_tmpa_dim
9339   {
9340     \l_@@_y_initial_dim - \l_@@_y_final_dim +
9341     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9342   }
9343   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9344   \group_begin:
9345   \pgfsetlinewidth { 1.1 \arrayrulewidth }
9346   \@@_set_CTarc:o \l_@@_rules_color_tl
9347   \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`.

```

9348   \seq_map_inline:Nn \g_@@_cols_vlism_seq
9349   {
9350     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
9351     {
9352       \int_compare:nNnT
9353       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9354       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9355         \@@_qpoint:n { col - ##1 }
9356         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9357         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9358         \pgfusepathqstroke
9359       }
9360     }
9361   }

```

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.

```

9362 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9363 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9364 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9365 {
9366   \bool_lazy_and:nnTF
9367   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9368   {
9369     \int_compare_p:nNn
9370     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9371   {
9372     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9373     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9374     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9375     \pgfusepathqstroke
9376   }
9377   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9378 }

```

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.

```

9379 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9380 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9381 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9382 {
9383   \bool_lazy_and:nnTF
9384   { \int_compare_p:nNn { ##1 } > \c_zero_int }
9385   {
9386     \int_compare_p:nNn
9387     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9388   {
9389     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9390 \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

9391 \dim_set:Nn \l_tmpa_dim
9392 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9393 \str_case:nn { #1 }
9394 {
9395   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9396   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9397   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9398   }
9399   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

9400 \dim_set:Nn \l_tmpb_dim
9401 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9402 \str_case:nn { #2 }
9403 {
9404   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9405   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9406   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9407 }
9408 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9409 \pgfusepathqstroke
9410 \group_end:
9411 }
9412 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9413 }

```

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).

```

9414   \str_if_empty:NF \l_@@_submatrix_name_str
9415   {
9416     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9417     \l_@@_x_initial_dim \l_@@_y_initial_dim
9418     \l_@@_x_final_dim \l_@@_y_final_dim
9419   }
9420   \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`.

```

9421   \begin { pgfscope }
9422   \pgftransformshift
9423   {
9424     \pgfpoint
9425     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9426     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9427   }
9428   \str_if_empty:NTF \l_@@_submatrix_name_str
9429   { \@@_node_left:nn #1 { } }
9430   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9431   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9432   \pgftransformshift
9433   {
9434     \pgfpoint
9435     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9436     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9437   }
9438   \str_if_empty:NTF \l_@@_submatrix_name_str
9439   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9440   {
9441     \@@_node_right:nnnn #2
9442     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9443   }

```

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`.

```

9444   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9445   \flag_clear_new:N \l_@@_code_flag
9446   \l_@@_code_tl
9447 }

```

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-lj* 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`.

```

9448 \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.

```
9449 \cs_new:Npn \@@_pgfpointanchor:n #1
9450 { \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`).

```
9451 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9452 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9453 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9454 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9455 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9456 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9457 { \@@_pgfpointanchor_ii:n { #1 } }
9458 }
```

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`).

```
9459 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9460 { \@@_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`.

```
9461 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
9462 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9463 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9464 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9465 { \@@_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`).

```
9466 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9467 }
```

The following function is for the case when the name contains an hyphen.

```
9468 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9469 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9470 \@@_env:
9471 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9472 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9473 }
```

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.

```

9474 \tl_const:Nn \c_@@_integers_alist_tl
9475 {
9476   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9477   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9478   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9479   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9480 }

9481 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9482 {

```

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`.

```

9483 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9484 {
9485   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9486 \@@_env: -
9487 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9488 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9489 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9490 }
9491 {
9492   \str_if_eq:eeTF { #1 } { last }
9493   {
9494     \flag_raise:N \l_@@_code_flag
9495     \@@_env: -
9496     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9497     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9498     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9499   }
9500   { #1 }
9501 }
9502 }

```

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`).

```

9503 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9504 {
9505   \pgfnode
9506   { rectangle }
9507   { east }
9508   {
9509     \nullfont
9510     $ % $
9511     \@@_color:o \l_@@_delimiters_color_tl
9512     \left #1
9513     \vcenter
9514     {
9515       \nullfont
9516       \hrule \@height \l_tmpa_dim
9517       \@depth \c_zero_dim

```

```

9518             \@width \c_zero_dim
9519         }
9520         \right .
9521         $ % $
9522     }
9523     { #2 }
9524     { }
9525 }

```

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.

```

9526 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9527 {
9528     \pgfnode
9529     { rectangle }
9530     { west }
9531     {
9532         \nullfont
9533         $ % $
9534         \colorlet { current-color } { . }
9535         \@@_color:o \l_@@_delimiters_color_tl
9536         \left .
9537         \vcenter
9538         {
9539             \nullfont
9540             \hrule \@height \l_tmpa_dim
9541                 \@depth \c_zero_dim
9542                 \@width \c_zero_dim
9543         }
9544         \right #1
9545         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9546         ^ { \color { current-color } \smash { #4 } }
9547         $ % $
9548     }
9549     { #2 }
9550     { }
9551 }

```

### 33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9552 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9553 {
9554     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9555     \ignorespaces
9556 }
9557 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9558 {
9559     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9560     \ignorespaces
9561 }
9562 \keys_define:nn { nicematrix / Brace }
9563 {
9564     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9565     left-shorten .value_forbidden:n = true ,
9566     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,

```

```

9567 right-shorten .value_forbidden:n = true ,
9568 shorten .meta:n = { left-shorten , right-shorten } ,
9569 shorten .value_forbidden:n = true ,
9570 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9571 color .tl_set:N = \l_tmpa_tl ,
9572 color .value_required:n = true ,
9573 unknown .code:n =
9574   \@@_unknown_key:nn
9575   { nicematrix / Brace }
9576   { Unknown~key~for~Brace }
9577 }

```

#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`.

```

9578 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9579 {
9580   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9581   \@@_compute_i_j:nn { #1 } { #2 }
9582   \bool_lazy_or:nnTF
9583     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9584     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9585     {
9586       \str_if_eq:eeTF { #5 } { under }
9587       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9588       { \@@_error:nn { Construct~too~large } { \OverBrace } }
9589     }
9590   {
9591     \tl_clear:N \l_tmpa_tl
9592     \keys_set:nn { nicematrix / Brace } { #4 }
9593     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9594     \pgfpicture
9595     \pgfrememberpicturerepositiononpagetrue
9596     \pgf@relevantforpicturesizefalse
9597     \bool_if:NT \l_@@_brace_left_shorten_bool
9598     {
9599       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9600       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9601       {
9602         \cs_if_exist:cT
9603           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9604           {
9605             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9606
9607             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9608             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9609           }
9610       }
9611     }
9612     \bool_lazy_or:nnT
9613       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9614       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9615       {
9616         \@@_qpoint:n { col - \l_@@_first_j_tl }
9617         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9618       }
9619     \bool_if:NT \l_@@_brace_right_shorten_bool
9620     {
9621       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9622       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9623       {
9624         \cs_if_exist:cT

```

```

9625         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9626         {
9627             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9628             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9629             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9630         }
9631     }
9632 }
9633 \bool_lazy_or:nnT
9634 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9635 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9636 {
9637     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9638     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9639 }
9640 \pgfset { inner~sep = \c_zero_dim }
9641 \str_if_eq:eeTF { #5 } { under }
9642 { \@@_underbrace_i:n { #3 } }
9643 { \@@_overbrace_i:n { #3 } }
9644 \endpgfpicture
9645 }
9646 \group_end:
9647 }

```

The argument is the text to put above the brace.

```

9648 \cs_new_protected:Npn \@@_overbrace_i:n #1
9649 {
9650     \@@_qpoint:n { row - \l_@@_first_i_tl }
9651     \pgftransformshift
9652     {
9653         \pgfpoint
9654         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9655         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9656     }
9657     \pgfnode
9658     { rectangle }
9659     { south }
9660     {
9661         \vtop
9662         {
9663             \group_begin:
9664             \everycr { }
9665             \halign
9666             {
9667                 \hfil ## \hfil \crcr
9668                 \bool_if:NTF \l_@@_tabular_bool
9669                 { \begin { tabular } { c } #1 \end { tabular } }
9670                 { $ \begin { array } { c } #1 \end { array } $ }
9671                 \cr
9672                 $ % $
9673                 \overbrace
9674                 {
9675                     \hbox_to_wd:nn
9676                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9677                     { }
9678                 }
9679                 $ % $
9680                 \cr
9681             }
9682             \group_end:
9683         }
9684     }
9685     { }
9686     { }

```

```
9687 }
```

The argument is the text to put under the brace.

```
9688 \cs_new_protected:Npn \@@_underbrace_i:n #1
9689 {
9690   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9691   \pgftransformshift
9692   {
9693     \pgfpoint
9694     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9695     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9696   }
9697   \pgfnode
9698   { rectangle }
9699   { north }
9700   {
9701     \group_begin:
9702     \everycr { }
9703     \vbox
9704     {
9705       \halign
9706       {
9707         \hfil ## \hfil \crcr
9708         $ % $
9709         \underbrace
9710         {
9711           \hbox_to_wd:nn
9712           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9713           { }
9714         }
9715         $ % $
9716         \cr
9717         \bool_if:NTF \l_@@_tabular_bool
9718         { \begin { tabular } { c } #1 \end { tabular } }
9719         { $ \begin { array } { c } #1 \end { array } $ }
9720         \cr
9721       }
9722     }
9723     \group_end:
9724   }
9725   { }
9726   { }
9727 }
```

## 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`).

```
9728 \AddToHook { package / tikz / after }
9729 {
9730   \tikzset
9731   {
9732     nicematrix / brace / .style =
9733     {
```

```

9734         decoration = { brace , raise = -0.15 em } ,
9735         decorate ,
9736     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9737     nicematrix / mirrored-brace / .style =
9738     {
9739         nicematrix / brace ,
9740         decoration = mirror ,
9741     }
9742 }
9743 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9744 \keys_define:nn { nicematrix / Hbrace }
9745 {
9746     color .code:n = ,
9747     horizontal-label .code:n = ,
9748     horizontal-labels .code:n = ,
9749     shorten .code:n = ,
9750     shorten-start .code:n = ,
9751     shorten-end .code:n = ,
9752     shorten+ .code:n = ,
9753     shorten-start+ .code:n = ,
9754     shorten-end+ .code:n = ,
9755     shorten~+ .code:n = ,
9756     shorten-start~+ .code:n = ,
9757     shorten-end~+ .code:n = ,
9758     brace-shift .code:n = ,
9759     brace-shift+ .code:n = ,
9760     brace-shift~+ .code:n = ,
9761     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9762 }

```

Here we need an “fully expandable” command.

```

9763 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9764 {
9765     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9766     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9767     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9768 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9769 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9770 {
9771     \int_compare:nNnTF \c@iRow < { 2 }
9772     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9773     \str_if_eq:nnTF { #2 } { * }
9774     {
9775         \bool_set_true:N \l_@@_nullify_dots_bool
9776         \Ldots
9777         [
9778             line-style = nicematrix / brace ,
9779             #1 ,
9780             up =
9781             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9782         ]
9783     }

```

```

9784     {
9785         \Hdotsfor
9786         [
9787             line-style = nicematrix / brace ,
9788             #1 ,
9789             up =
9790             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9791         ]
9792         { #2 }
9793     }
9794 }
9795 {
9796     \str_if_eq:nnTF { #2 } { * }
9797     {
9798         \bool_set_true:N \l_@@_nullify_dots_bool
9799         \Ldots
9800         [
9801             line-style = nicematrix / mirrored-brace ,
9802             #1 ,
9803             down =
9804             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9805         ]
9806     }
9807     {
9808         \Hdotsfor
9809         [
9810             line-style = nicematrix / mirrored-brace ,
9811             #1 ,
9812             down =
9813             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9814         ]
9815         { #2 }
9816     }
9817 }
9818 \keys_set:nn { nicematrix / Hbrace } { #1 }
9819 }

```

```

9820 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9821 {
9822     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9823     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9824     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9825 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9826 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9827 {
9828     \int_compare:nNnTF \c_jCol < { 2 }
9829     {
9830         \str_if_eq:nnTF { #2 } { * }
9831         {
9832             \bool_set_true:N \l_@@_nullify_dots_bool
9833             \Vdots
9834             [
9835                 Vbrace ,
9836                 line-style = nicematrix / mirrored-brace ,
9837                 #1 ,
9838                 down =
9839                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9840             ]
9841         }
9842     }

```

```

9843         \Vdotsfor
9844         [
9845             Vbrace ,
9846             line-style = nicematrix / mirrored-brace ,
9847             #1 ,
9848             down =
9849             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9850         ]
9851     { #2 }
9852 }
9853 }
9854 {
9855     \str_if_eq:nnTF { #2 } { * }
9856     {
9857         \bool_set_true:N \l_@@_nullify_dots_bool
9858         \Vdots
9859         [
9860             Vbrace ,
9861             line-style = nicematrix / brace ,
9862             #1 ,
9863             up =
9864             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9865         ]
9866     }
9867     {
9868         \Vdotsfor
9869         [
9870             Vbrace ,
9871             line-style = nicematrix / brace ,
9872             #1 ,
9873             up =
9874             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9875         ]
9876         { #2 }
9877     }
9878 }
9879 \keys_set:nn { nicematrix / Hbrace } { #1 }
9880 }

```

## 35 The command TikzEveryCell

```

9881 \bool_new:N \l_@@_not_empty_bool
9882 \bool_new:N \l_@@_empty_bool
9883
9884 \keys_define:nn { nicematrix / TikzEveryCell }
9885 {
9886     not-empty .code:n =
9887         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9888         { \bool_set_true:N \l_@@_not_empty_bool }
9889         { \@@_error:n { detection-of-empty-cells } } ,
9890     not-empty .value_forbidden:n = true ,
9891     empty .code:n =
9892         \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9893         { \bool_set_true:N \l_@@_empty_bool }
9894         { \@@_error:n { detection-of-empty-cells } } ,
9895     empty .value_forbidden:n = true ,
9896     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9897 }
9898

```

```

9899
9900 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9901 {
9902   \IfPackageLoadedTF { tikz }
9903   {
9904     \group_begin:
9905     \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.

```

9906     \tl_set:Nn \l_tmpa_tl { { #2 } }
9907     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9908     { \@@_for_a_block:nnnnn ##1 }
9909     \@@_all_the_cells:
9910     \group_end:
9911   }
9912   { \@@_error:n { TikzEveryCell~without~tikz } }
9913 }
9914
9915
9916 \cs_new_protected:Nn \@@_all_the_cells:
9917 {
9918   \int_step_inline:nn \c@iRow
9919   {
9920     \int_step_inline:nn \c@jCol
9921     {
9922       \cs_if_exist:cF { cell - ##1 - #####1 }
9923       {
9924         \clist_if_in:Nef \l_@@_corners_cells_clist
9925         { ##1 - #####1 }
9926         {
9927           \bool_set_false:N \l_tmpa_bool
9928           \cs_if_exist:cTF
9929           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9930           {
9931             \bool_if:NF \l_@@_empty_bool
9932             { \bool_set_true:N \l_tmpa_bool }
9933           }
9934           {
9935             \bool_if:NF \l_@@_not_empty_bool
9936             { \bool_set_true:N \l_tmpa_bool }
9937           }
9938           \bool_if:NT \l_tmpa_bool
9939           {
9940             \@@_block_tikz:nnnnn
9941             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9942           }
9943         }
9944       }
9945     }
9946   }
9947 }
9948
9949 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9950 {
9951   \bool_if:NF \l_@@_empty_bool
9952   {
9953     \@@_block_tikz:nnnnn
9954     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9955   }
9956   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9957 }
9958
9959 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn

```

```

9960 {
9961   \int_step_inline:nnn { #1 } { #3 }
9962   {
9963     \int_step_inline:nnn { #2 } { #4 }
9964     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9965   }
9966 }

```

## 36 The key draw-trees-in-col

```

9967 \cs_new_protected:Npn \@@_draw_trees:
9968 {
9969   \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
9970   \dim_zero_new:N \l_@@_em_dim
9971   \dim_set:Nn \l_@@_em_dim { 1 em }
9972   \dim_zero_new:N \l_@@_ex_dim
9973   \dim_set:Nn \l_@@_ex_dim { 1 ex }
9974   \pgfpicture
9975   \pgfrememberpicturepositiononpagetrue
9976   \pgf@relevantforpicturesizefalse
9977   \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
9978     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
9979   \@@_color:o \l_@@_trees_color_tl
9980   \pgfsetcornersarced
9981     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
9982   \clist_map_function:NN \g_@@_col_with_trees_clist
9983     \@@_draw_trees_in_col:n
9984   \clist_gclear:N \g_@@_col_with_trees_clist
9985   \endpgfpicture
9986 }

9987 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
9988 {

```

The argument is provided by curryfication.

```

9989   \int_compare:nNnTF { #1 } > \c@jCol
9990     { \@@_error:nn { Col-outside-tabular~in~trees } }
9991     {
9992       \int_compare:nNnTF { #1 } = \c@jCol
9993         { \@@_error:nn { Last-col-in~trees } }
9994         \@@_draw_trees_in_col_i:n
9995     }
9996   { #1 }
9997 }

```

```

9998 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
9999 {
10000   \int_set:Nn \l_tmpa_int { 1 }
10001   \int_step_inline:nn { \c@iRow + 1 }
10002   {
10003     \cs_if_exist:cT
10004       { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10005       {
10006         \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10007         {

```

Now, you will, potentially, draw a tree.

```

10008       \@@_draw_tree:nee
10009       { #1 }
10010       { \int_use:N \l_tmpa_int }
10011       { \int_eval:n { ##1 - 1 } }
10012     }

```

```

10013         \int_set:Nn \l_tmpa_int { ##1 }
10014     }
10015 }
10016 \int_compare:nNnT \c@iRow > \l_tmpa_int
10017 {
10018     \@@_draw_tree:nee
10019     { #1 }
10020     { \int_use:N \l_tmpa_int }
10021     { \int_eval:n { \c@iRow } }
10022 }
10023 }

```

#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.

```

10024 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10025 {

```

\l\_tmpa\_dim will be the  $x$ -value of the vertical rule that we will draw.

```

10026     \pgfpointanchor { \@@_env: - #1 } { 5 }
10027     \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.

```

10028     \bool_set_false:N \l_tmpa_bool
10029     \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10030     {
10031         \cs_if_exist:cT
10032         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10033     {
10034         \pgfpointanchor
10035         { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10036         { base~west }
10037         \pgfpathmoveto
10038         {
10039             \pgfpoint
10040             { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }
10041             { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10042         }
10043         \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10044         \bool_if:NF \l_tmpa_bool
10045         {
10046             \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10047             \pgfpathlineto
10048             { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10049             \bool_set_true:N \l_tmpa_bool
10050         }
10051         \pgfusepath { stroke }
10052     }
10053 }
10054 }
10055 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

## 37 The key create-blocks-in-col

```

10056 \cs_new_protected:Npn \@@_create_blocks_in_col:
10057 {
10058     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10059     \clist_map_inline:Nn \g_@@_cbic_clist
10060     {
10061         \cs_set:cpn
10062         {

```

```

10063         pgf @ sh @ ns @ \@@_env:
10064         - \int_eval:n { \c@iRow + 1 } - ##1 }
10065     { rien }

\l_tmpa_int will be the first row of the block being created.
10066     \int_set:Nn \l_tmpa_int { 1 }
10067     \int_step_inline:nn { \c@iRow + 1 }
10068     {
10069         \cs_if_exist:cT
10070         { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10071         {
10072             \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10073             {
10074                 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10075                 {
10076                     { \int_use:N \l_tmpa_int }
10077                     { ##1 }
10078                     { \int_eval:n { #####1 - 1 } }
10079                     { ##1 }
10080                     { }
10081                 }
10082             }
10083             \int_set:Nn \l_tmpa_int { #####1 }
10084         }
10085     }
10086     \clist_gclear:N \g_@@_cbic_clist
10087 }
10088

```

## 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.

```

10089 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10090 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10091 \NewDocumentCommand \@@_ShowCellNames { }
10092 {
10093     \bool_if:NT \l_@@_in_code_after_bool
10094     {
10095         \pgfpicture
10096         \pgfrememberpicturepositiononpagetrue
10097         \pgf@relevantforpicturesizefalse
10098         \pgfpathrectanglecorners
10099         { \@@_qpoint:n { 1 } }
10100         { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10101         \pgfsetfillopacity { 0.75 }
10102         \pgfsetfillcolor { white }
10103         \pgfusepathqfill
10104         \endpgfpicture
10105     }
10106     \dim_gzero_new:N \g_@@_tmpc_dim
10107     \dim_gzero_new:N \g_@@_tmpd_dim
10108     \dim_gzero_new:N \g_@@_tmpe_dim
10109     \int_step_inline:nn \c@iRow
10110     {
10111         \bool_if:NTF \l_@@_in_code_after_bool
10112         {

```

```

10113     \pgfpicture
10114     \pgfrememberpicturepositiononpagetrue
10115     \pgf@relevantforpicturesizefalse
10116   }
10117   { \begin { pgfpicture } }
10118   \@@_qpoint:n { row - ##1 }
10119   \dim_set_eq:Nn \l_tmpa_dim \pgf@y
10120   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10121   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10122   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10123   \bool_if:NTF \l_@@_in_code_after_bool
10124     { \endpgfpicture }
10125   { \end { pgfpicture } }
10126   \int_step_inline:nn \c@jCol
10127   {
10128     \hbox_set:Nn \l_tmpa_box
10129     {
10130       \normalfont \Large \sffamily \bfseries
10131       \bool_if:NTF \l_@@_in_code_after_bool
10132         { \color { red } }
10133         { \color { red ! 50 } }
10134       ##1 - ####1
10135     }
10136     \bool_if:NTF \l_@@_in_code_after_bool
10137     {
10138       \pgfpicture
10139       \pgfrememberpicturepositiononpagetrue
10140       \pgf@relevantforpicturesizefalse
10141     }
10142     { \begin { pgfpicture } }
10143     \@@_qpoint:n { col - ####1 }
10144     \dim_gset_eq:Nn \g_@@_tmpc_dim \pgf@x
10145     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10146     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10147     \dim_gset_eq:Nn \g_@@_tmpe_dim \pgf@x
10148     \bool_if:NTF \l_@@_in_code_after_bool
10149       { \endpgfpicture }
10150     { \end { pgfpicture } }
10151     \fp_set:Nn \l_tmpa_fp
10152     {
10153       \fp_min:nn
10154       {
10155         \fp_min:nn
10156         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10157         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10158       }
10159       { 1.0 }
10160     }
10161     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10162     \pgfpicture
10163     \pgfrememberpicturepositiononpagetrue
10164     \pgf@relevantforpicturesizefalse
10165     \pgftransformshift
10166     {
10167       \pgfpoint
10168       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10169       \g_tmpa_dim
10170     }
10171     \pgfnode
10172     { rectangle }
10173     { center }
10174     { \box_use:N \l_tmpa_box }
10175     { }

```

```

10176         { }
10177     \endpgfpicture
10178 }
10179 }
10180 }

```

### 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.

```

10181 \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.

```

10182 \bool_new:N \g_@@_footnote_bool

10183 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10184 {
10185     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10186     but-that-key-is-unknown. \\
10187     It-will-be-ignored. \\
10188     For-a-list-of-the-available-keys,~type-H~<return>.
10189 }
10190 {
10191     The-available-keys-are~(in-alphabetic-order):~
10192     footnote,~
10193     footnotehyper,~
10194     messages-for-Overleaf,~
10195     renew-dots~and~
10196     renew-matrix.
10197 }

10198 \keys_define:nn { nicematrix }
10199 {
10200     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10201     renew-dots .value_forbidden:n = true ,
10202     renew-matrix .code:n = \@@_renew_matrix: ,
10203     renew-matrix .value_forbidden:n = true ,
10204     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10205     footnote .bool_set:N = \g_@@_footnote_bool ,
10206     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10207     unknown .code:n = \@@_error:n { Unknown~key~for~package }
10208 }
10209 \ProcessKeyOptions

10210 \@@_msg_new:nn { footnote~with~footnotehyper~package }
10211 {
10212     You-can't-use-the-option~'footnote'~because-the-package~
10213     footnotehyper~has-already-been-loaded.~
10214     If-you-want,~you-can-use-the-option~'footnotehyper'~and-the-footnotes~
10215     within~the~environments~of~nicematrix~will-be~extracted~with~the~tools~
10216     of~the~package~footnotehyper.\\
10217     The~package~footnote~won't-be-loaded.
10218 }

```

```

10219 \@@_msg_new:nn { footnotehyper~with~footnote~package }
10220 {
10221   You~can't~use~the~option~'footnotehyper'~because~the~package~
10222   footnote~has~already~been~loaded.~
10223   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10224   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10225   of~the~package~footnote.\\
10226   The~package~footnotehyper~won't~be~loaded.
10227 }

```

```

10228 \bool_if:NT \g_@@_footnote_bool
10229 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10230   \IfClassLoadedTF { beamer }
10231   { \bool_set_false:N \g_@@_footnote_bool }
10232   {
10233     \IfPackageLoadedTF { footnotehyper }
10234     { \@@_error:n { footnote~with~footnotehyper~package } }
10235     { \usepackage { footnote } }
10236   }
10237 }

```

```

10238 \bool_if:NT \g_@@_footnotehyper_bool
10239 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10240   \IfClassLoadedTF { beamer }
10241   { \bool_set_false:N \g_@@_footnote_bool }
10242   {
10243     \IfPackageLoadedTF { footnote }
10244     { \@@_error:n { footnotehyper~with~footnote~package } }
10245     { \usepackage { footnotehyper } }
10246   }
10247   \bool_set_true:N \g_@@_footnote_bool
10248 }

```

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.

```

10249 \bool_new:N \l_@@_underscore_loaded_bool
10250 \IfPackageLoadedT { underscore }
10251 { \bool_set_true:N \l_@@_underscore_loaded_bool }
10252 \AtBeginDocument
10253 {
10254   \bool_if:NF \l_@@_underscore_loaded_bool
10255   {
10256     \IfPackageLoadedT { underscore }
10257     { \@@_error:n { underscore~after~nicematrix } }
10258   }
10259 }

```

## 41 Compatibility with threeparttable

```
10260 \AtBeginDocument
10261 {
10262   \IfPackageLoadedT { threeparttable }
10263   {
10264     \AddToHook { env / threeparttable / begin }
10265     {
10266       \TPT@hookin { NiceTabular }
10267       \TPT@hookin { NiceTabular* }
10268       \TPT@hookin { NiceTabularX }
10269     }
10270   }
10271 }
```

## 42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent “additive syntax” for that key. Of course, in that case, the last character of the name of the key is +.

#1 is a clist of names of sets of keys and #2 is the error message to send.

```
10272 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10273 {
10274   \str_if_eq:eeTF
10275   { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10276   { + }
10277   {
10278     \str_set:Ne \l_tmpa_str
10279     { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10280     \bool_set_false:N \l_tmpa_bool
10281     \clist_map_inline:nn { #1 }
10282     {
10283       \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10284       {
10285         \@@_error:n { key~without~+~exists }
10286         \bool_set_true:N \l_tmpa_bool
10287         \clist_map_break:
10288       }
10289     }
10290     \bool_if:NF \l_tmpa_bool
10291     {
10292       \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10293       \@@_unknown_key_i:nn { #1 } { #2 }
10294     }
10295   }
10296   { \@@_unknown_key_i:nn { #1 } { #2 } }
10297 }
```

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.

```
10298 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10299 {
10300   \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10301   \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10302   \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10303   \bool_set_false:N \l_tmpa_bool
```

```

10304 \clist_map_inline:nn { #1 }
10305 {
10306     \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10307     {
10308         \@@_error:n { key~with~normal~form~exists }
10309         \bool_set_true:N \l_tmpa_bool
10310         \clist_map_break:
10311     }
10312 }
10313 \bool_if:NF \l_tmpa_bool
10314 {
10315     \@@_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).

```

10316     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10317     { \msg_info:nn { nicematrix } { #2~+ } }
10318 }
10319 }
10320 \@@_msg_new:nn { key~without~+~exists }
10321 {
10322     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10323     additive~syntax~(with~+=).\
10324     It~will~be~ignored.\
10325 }
10326 \@@_msg_new:nn { key~with~normal~form~exists }
10327 {
10328     No~key~'\l_keys_key_str'.\
10329     It~will~be~ignored.\
10330     Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10331 }
10332 \str_const:Ne \c_@@_available_keys_str
10333 {
10334     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
10335     { For~a~list~of~the~available~keys,~type~H~<return>. }
10336 }
10337 \seq_new:N \g_@@_types_of_matrix_seq
10338 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10339 {
10340     NiceMatrix ,
10341     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10342 }
10343 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10344 { \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`.

```

10345 \cs_new_protected:Npn \@@_err_too_many_cols:
10346 {
10347     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10348     { \@@_fatal:nn { too-many-cols-for-array } }
10349     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10350     { \@@_fatal:n { too-many-cols-for-matrix } }
10351     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10352     { \@@_fatal:n { too-many-cols-for-matrix } }
10353     \bool_if:NF \l_@@_last_col_without_value_bool
10354     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10355 }

```

The following command must *not* be protected since it's used in an error message.

```

10356 \cs_new:Npn \@@_message_hdotsfor:
10357 {
10358   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
10359   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
10360     \token_to_str:N \Hbrace \ is~incorrect. }
10361 }

10362 \cs_new_protected:Npn \@@_Hline_in_cell:
10363 { \@@_fatal:n { Misuse-of-Hline } }

10364 \@@_msg_new:nn { Misuse-of-Hline }
10365 {
10366   Misuse-of~Hline. \\
10367   Error~in~your~row~ \int_eval:N \c@iRow . \\
10368   \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must~be~used~only~
10369   at~the~beginning~of~a~row.\\
10370   That~error~is~fatal.
10371 }

10372 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
10373 {
10374   Incompatible~options.\\
10375   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
10376   The~output~will~not~be~reliable.
10377 }

10378 \@@_msg_new:nn { Body-alone }
10379 {
10380   \token_to_str:N \Body\ alone. \\
10381   You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
10382   That~error~is~fatal.
10383 }

10384 \@@_msg_new:nn { Bad-use-of-CodeBefore }
10385 {
10386   Bad-use-of~\token_to_str:N \CodeBefore. \\
10387   \token_to_str:N \CodeBefore\ must~be~used~only~at~the~beginning~of~
10388   the~environment.\\
10389   That~error~is~fatal.
10390 }

10391 \@@_msg_new:nn { NiceTabularX~probably~required }
10392 {
10393   Incorrect~syntax.\\
10394   You~probably~want~to~use~\{NiceTabularX\}~whose~first~argument~is~the~
10395   disered~width~of~the~tabular.\\
10396   That~error~is~fatal.
10397 }

10398 \@@_msg_new:nn { cellcolor~in~Block }
10399 {
10400   Bad-use-of~\token_to_str:N \cellcolor \\
10401   You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
10402   \bool_if:NTF \l_@@_amp_in_blocks_bool
10403   { (but~you~could~use~it~in~a~sub~block~since~'&~in~blocks'~is~in~force) }
10404   { (it's~possible~in~a~sub~block~when~'&~in~blocks'~is~in~force) }
10405   .~Here,~you~should~use~the~key~'fill'~of~the~block.\\
10406   That~command~will~be~ignored.
10407 }

10408 \@@_msg_new:nn { rowcolor~in~Block }
10409 {
10410   Bad-use-of~\token_to_str:N \rowcolor \\
10411   You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
10412   However,~it's~possible~to~color~the~block~with~its~key~'fill'.\\
10413   That~command~will~be~ignored.
10414 }

```

```

10415 \@@_msg_new:nn { key~color~inside }
10416 {
10417   Deleted~key.\\
10418   The~key~'color~inside'~(and~its~alias~'colortbl~like')~has~been~deleted~in
10419   ~'nicematrix'~and~must~not~be~used.\\
10420   This~error~is~fatal.
10421 }
10422 \@@_msg_new:nn { Invalid~argument~for~w }
10423 {
10424   Invalid~argument~for~w.\\
10425   You~have~used~the~type~of~alignment~'#1'~for~your~column~'w'~
10426   but~only~'c',~'r',~'l'~and~'s'~are~allowed~in~a~column~'w'.~
10427   If~you~go~on,~'c'~will~be~used.
10428 }
10429 \@@_msg_new:nn { invalid~weight }
10430 {
10431   Unknown~key.\\
10432   The~key~'\l_keys_key_str'~of~your~column~X~is~unknown~and~will~be~ignored.~
10433   The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
10434   \IfPackageLoadedTF { varwidth }
10435     { (since~'varwidth'~is~loaded)~}
10436     { (if~you~load~'varwidth')~}
10437   and~real~numbers~for~the~weight~of~the~X~column.
10438 }
10439 \@@_msg_new:nn { last~col~not~used }
10440 {
10441   Column~not~used.\\
10442   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
10443   in~your~\@@_full_name_env: .~
10444   However,~you~can~go~on.
10445 }
10446 \@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
10447 {
10448   Too~many~columns.\\
10449   In~the~row~ \int_eval:n { \c@iRow },~
10450   you~try~to~use~more~columns~
10451   than~allowed~by~your~ \@@_full_name_env: .
10452   \@@_message_hdotsfor: \
10453   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10454   (plus~the~exterior~columns).\\
10455   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10456   This~error~is~fatal.
10457 }
10458 \@@_msg_new:nn { too~many~cols~for~matrix }
10459 {
10460   Too~many~columns.\\
10461   In~the~row~ \int_eval:n { \c@iRow } ,~
10462   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10463   \@@_message_hdotsfor: \
10464   Recall~that~the~maximal~number~of~columns~for~a~matrix~
10465   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10466   LaTeX~counter~'MaxMatrixCols'.~
10467   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10468   (use~ \token_to_str:N \setcounter \ to~change~that~value).\\
10469   But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10470   This~error~is~fatal.
10471 }
10472 \@@_msg_new:nn { too~many~cols~for~array }
10473 {
10474   Too~many~columns.\\

```

```

10475 In~the~row~ \int_eval:n { \c@iRow } ,~
10476 ~you~try~to~use~more~columns~than~allowed~by~your~
10477 \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10478 \int_use:N \g_@@_static_num_of_col_int \
10479 \bool_if:nT
10480 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10481 { (plus~the~exterior~ones)~}
10482 since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
10483 But,~maybe,~you~have~forgotten~a~\token_to_str:N \\. \\
10484 This~error~is~fatal.
10485 }
10486 \@@_msg_new:nn { columns~not~used }
10487 {
10488 Columns~not~used.\\
10489 The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10490 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10491 columns~but~you~only~used~ \int_use:N \c@jCol .\\
10492 The~columns~you~did~not~used~won't~be~created.\\
10493 You~won't~have~similar~warning~till~the~end~of~the~document.
10494 }
10495 }
10496 \@@_msg_new:nn { Bad~use~of~NiceTabularNotes }
10497 {
10498 Bad~use~of~\token_to_str:N \NiceTabularNotes \\
10499 \token_to_str:N~\NiceTabularNotes\ should~be~used~only~
10500 after~at~tabular~which~uses~`notes/no-print`. \\
10501 That~command~will~be~ignored.
10502 }
10503 \@@_msg_new:nn { empty~preamble }
10504 {
10505 Empty~preamble.\\
10506 The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
10507 This~error~is~fatal.
10508 }
10509 \@@_msg_new:nn { in~first~col }
10510 {
10511 Erroneous~use.\\
10512 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
10513 That~command~will~be~ignored.\\
10514 You~can~try~to~delete~the~key~'first-col'.
10515 }
10516 \@@_msg_new:nn { in~last~col }
10517 {
10518 Erroneous~use.\\
10519 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
10520 That~command~will~be~ignored.\\
10521 You~can~try~to~delete~the~key~'last-col'.
10522 }
10523 \@@_msg_new:nn { in~first~row }
10524 {
10525 Erroneous~use.\\
10526 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
10527 That~command~will~be~ignored.\\
10528 You~can~try~to~delete~the~key~'first-row'.
10529 }
10530 \@@_msg_new:nn { in~last~row }
10531 {
10532 Erroneous~use.\\
10533 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
10534 That~command~will~be~ignored.\\
10535 You~can~try~to~delete~the~key~'last-row'.

```

```

10536 }
10537 \@@_msg_new:nn { TopRule~without~booktabs }
10538 {
10539   Erroneous~use.\\
10540   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
10541   You~should~load~'booktabs'~(before~or~after~'nicematrix').\\
10542   That~command~will~be~ignored.
10543 }
10544 \@@_msg_new:nn{ rotate~in~p~col }
10545 {
10546   \token_to_str:N \rotate\ forbidden.\\
10547   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10548   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10549   If~you~go~on,~maybe~you~won't~have~the~expected~output.
10550 }
10551 \@@_msg_new:nn { TopRule~without~tikz }
10552 {
10553   Erroneous~use.\\
10554   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.\\
10555   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
10556   \IfPackageLoadedF { booktabs }
10557   { You~should~also~load~'booktabs'.\\ }
10558   That~command~will~be~ignored.
10559 }
10560 \@@_msg_new:nn { caption~outside~float }
10561 {
10562   Key~caption~forbidden.\\
10563   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10564   environment~(such~as~\{table\}).~This~key~will~be~ignored.
10565 }
10566 \@@_msg_new:nn { short~caption~without~caption }
10567 {
10568   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10569   However,~your~'short~caption'~will~be~used~as~'caption'.
10570 }
10571 \@@_msg_new:nn { double~closing~delimiter }
10572 {
10573   Double~delimiter.\\
10574   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10575   delimiter.~This~delimiter~will~be~ignored.\\
10576   You~can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10577 }
10578 \@@_msg_new:nn { delimiter~after~opening }
10579 {
10580   Double~delimiter.\\
10581   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10582   delimiter.~That~delimiter~will~be~ignored.
10583 }
10584 \@@_msg_new:nn { bad~option~for~line~style }
10585 {
10586   Bad~line~style.\\
10587   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10588   is~'standard'.~That~key~will~be~ignored.\\
10589   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10590   before~or~after~'nicematrix'.\\
10591 }
10592 \@@_msg_new:nn { corners~with~no~cell~nodes }
10593 {
10594   Incompatible~keys.\\
10595   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~

```

```

10596     is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10597     is~to~speed~up~compilation).\\
10598     If~you~go~on,~that~key~will~be~ignored.
10599 }

10600 \@@_msg_new:nn { extra-nodes~with~no-cell-nodes }
10601 {
10602     Incompatible~keys.\\
10603     You~can't~create~'extra-nodes'~here~because~the~key~'no-cell-nodes'~
10604     is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10605     is~to~speed~up~compilation).\\
10606     If~you~go~on,~those~extra~nodes~won't~be~created.
10607 }

10608 \@@_msg_new:nn { Identical~notes~in~caption }
10609 {
10610     Identical~tabular~notes.\\
10611     You~can't~put~several~notes~with~the~same~content~in~
10612     \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\\
10613     If~you~go~on,~the~output~will~probably~be~erroneous.
10614 }

10615 \@@_msg_new:nn { tabularnote~below~the~tabular }
10616 {
10617     \token_to_str:N \tabularnote \ forbidden\\
10618     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10619     of~your~tabular~because~the~caption~will~be~composed~below~
10620     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10621     key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10622     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10623     no~similar~error~will~raised~in~this~document.
10624 }

10625 \@@_msg_new:nn { Unknown~key~for~rules }
10626 {
10627     Unknown~key.\\
10628     There~are~only~three~keys~available~here:~'width',~'color'~and~
10629     'fix-vertex'.\\
10630     Your~key~' \l_keys_key_str '~will~be~ignored.
10631 }

10632 \@@_msg_new:nn { Unknown~key~for~trees }
10633 {
10634     Unknown~key.\\
10635     There~are~only~three~keys~available~here:~width~color~and~
10636     rounded~corners.\\
10637     Your~key~' \l_keys_key_str '~will~be~ignored.
10638 }
10639 % \end{macrocode}
10640 %
10641 %
10642 % \begin{macrocode}
10643 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10644 {
10645     Unknown~key.\\
10646     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10647     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10648     and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10649     'horizontal-label(s)',~'shorten'~'shorten-end'~
10650     and~'shorten-start'.\\
10651     That~error~is~fatal.
10652 }

10653 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10654 {
10655     Unknown~key.\\
10656     There~is~only~two~keys~available~here:~

```

```

10657     'empty'~and~'not-empty'.\\
10658     Your~key~' \l_keys_key_str '~will~be~ignored.
10659 }

10660 \@@_msg_new:nn { Unknown~key~for~rotate }
10661 {
10662     Unknown~key.\\
10663     The~only~key~available~here~is~'c'.\\
10664     Your~key~' \l_keys_key_str '~will~be~ignored.
10665 }

10666 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10667 {
10668     Unknown~key.\\
10669     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10670     It~you~go~on,~you~will~probably~have~other~errors. \\
10671     \c_@@_available_keys_str
10672 }
10673 {
10674     The~available~keys~are~(in~alphabetic~order):~
10675     ccommand,~
10676     color,~
10677     command,~
10678     dotted,~
10679     letter,~
10680     multiplicity,~
10681     sep-color,~
10682     tikz,~and~total-width.
10683 }

10684 \@@_msg_new:nnn { Unknown~key~for~default~line }
10685 {
10686     Unknown~key.\\
10687     The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10688     It~you~go~on,~you~will~probably~have~other~errors. \\
10689     \c_@@_available_keys_str
10690 }
10691 {
10692     The~available~keys~are~(in~alphabetic~order):~
10693     color,~
10694     dotted,~
10695     multiplicity,~
10696     sep-color,~
10697     tikz,~and~total-width.
10698 }

10699 \@@_msg_new:nnn { Unknown~key~for~xdots }
10700 {
10701     Unknown~key.\\
10702     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10703     \c_@@_available_keys_str
10704 }
10705 {
10706     The~available~keys~are~(in~alphabetic~order):~
10707     'color',~
10708     'horizontal(s)-labels',~
10709     'inter',~
10710     'line-style',~
10711     'nullify',~
10712     'radius',~
10713     'shorten',~
10714     'shorten-end'~and~'shorten-start'.
10715 }

10716 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10717 {
10718     Unknown~key.\\

```

```

10719     As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
10720     (and-you-try-to-use~' \l_keys_key_str '~)\
10721     That-key-will-be-ignored.
10722 }

10723 \@@_msg_new:nn { Col-outside-tabular-in-trees }
10724 {
10725     Error-with~'draw-trees-in-col' \
10726     The-number-of-column~'#1'~is-outside-your-tabular~since-the-last-column~
10727     is~\int_use:N \c@jCol. \
10728     It-will-be-ignored.
10729 }

10730 \@@_msg_new:nn { Last-col-in-trees }
10731 {
10732     Error-with~'draw-trees-in-col' \
10733     You-can't-use~'draw-trees-in-col'~with-the-column~'#1'~
10734     because-it's-the-last-column-of-your-tabular. \
10735     It-will-be-ignored.
10736 }

10737 \@@_msg_new:nn { label-without-caption }
10738 {
10739     You-can't-use-the-key~'label'~in-your~\{NiceTabular\}~because~
10740     you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
10741 }

10742 \@@_msg_new:nn { W-warning }
10743 {
10744     Line~ \msg_line_number: .~The-cell-is-too-wide-for-your-column~'W'~
10745     (row~ \int_use:N \c@iRow ).
10746 }

10747 \@@_msg_new:nn { Construct-too-large }
10748 {
10749     Construct-too-large.\
10750     Your~command~ \token_to_str:N #1
10751     can't-be-drawn-because-your-matrix-is-too-small.\
10752     That-command-will-be-ignored.
10753 }

10754 \@@_msg_new:nn { underscore-after-nicematrix }
10755 {
10756     Problem-with~'underscore'.\
10757     The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
10758     You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
10759     ' \token_to_str:N \Cdots \token_to_str:N _
10760     \{ n \token_to_str:N \text \{ ~times \} \}' .
10761 }

10762 \@@_msg_new:nn { ampersand-in-light-syntax }
10763 {
10764     Ampersand-forbidden.\
10765     You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns~because~
10766     the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
10767 }

10768 \@@_msg_new:nn { double-backslash-in-light-syntax }
10769 {
10770     Double-backslash-forbidden.\
10771     You-can't-use~ \token_to_str:N \
10772     ~to-separate-rows~because-the-key~'light-syntax'~
10773     is-in-force.~You-must-use-the-character~' \l_@@_end_of_row_tl '~
10774     (set-by-the-key~'end-of-row').~This-error-is-fatal.
10775 }

10776 \@@_msg_new:nn { hlines-with-color }
10777 {
10778     Incompatible-keys.\

```

```

10779     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for-a~
10780     \token_to_str:N \Block \ when-the-key-'color'~or~'draw'~is-used.\\
10781     However,~you-can-put~several~commands~ \token_to_str:N \Block.\\
10782     Your-key-will-be-discarded.
10783 }
10784 \@@_msg_new:nn { bad-value-for-baseline }
10785 {
10786     Bad-value-for-baseline.\\
10787     The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10788     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
10789     \int_use:N \g_@@_row_total_int \ or-equal-to-'t',~'c'~or~'b'~or-of~
10790     the-form-'line-i'.\\
10791     A-value-of~1~will-be-used.
10792 }
10793 \@@_msg_new:nn { bad-value-for-baseline-line }
10794 {
10795     Bad-value-for-baseline-with-line.\\
10796     The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10797     valid.~The-number-of-the-line-must-be-between~1~and~
10798     \int_eval:n { \c@iRow + 1 } \\
10799     A-value-of-'line-1'~will-be-used.
10800 }
10801 \@@_msg_new:nn { detection-of-empty-cells }
10802 {
10803     Problem-with-'not-empty'\\
10804     For-technical-reasons,~you-must-activate~
10805     'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10806     in-order-to-use-the-key~' \l_keys_key_str '.\\
10807     That-key-will-be-ignored.
10808 }
10809 \@@_msg_new:nn { siunitx-not-loaded }
10810 {
10811     siunitx-not-loaded\\
10812     You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\\
10813     That-error-is-fatal.\\
10814     You-can-load-'siunitx'~with~\token_to_str:N \usepackage \{siunitx\},~
10815     before-or-after-'nicematrix'. \\
10816 }
10817 \@@_msg_new:nn { siunitx-too-old }
10818 {
10819     siunitx-too-old\\
10820     You-can't-use-the-columns-'S'~because-your-version-of-'siunitx'~
10821     is-too-old.~You-need-at-least-the-version-3.5.1~(2026/03/26).\\
10822     That-error-is-fatal.\\
10823 }
10824 \@@_msg_new:nn { Invalid-name }
10825 {
10826     Invalid-name.\\
10827     You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
10828     \SubMatrix \ of-your~ \@@_full_name_env: .\\
10829     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
10830     This-key-will-be-ignored.
10831 }
10832 \@@_msg_new:nn { Hbrace-not-allowed }
10833 {
10834     Command-not-allowed.\\
10835     You-can't-use-the-command~ \token_to_str:N #1
10836     because-you-have-not-loaded~
10837     \IfPackageLoadedTF { tikz }
10838     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10839     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }

```

```

10840 \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10841 That~command~will~be~ignored.
10842 }
10843 \@@_msg_new:nn { Vbrace~not~allowed }
10844 {
10845   Command~not~allowed.\\
10846   You~can't~use~the~command~ \token_to_str:N \Vbrace \
10847   because~you~have~not~loaded~TikZ~
10848   and~the~TikZ~library~'decorations.pathreplacing'.\\
10849   Use: ~\token_to_str:N \usepackage \{tikz\}~
10850   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10851   That~command~will~be~ignored.
10852 }
10853 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10854 {
10855   Wrong~line.\\
10856   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10857   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10858   number~is~not~valid.~It~will~be~ignored.
10859 }
10860 \@@_msg_new:nn { Impossible~delimiter }
10861 {
10862   Impossible~delimiter.\\
10863   It's~impossible~to~draw~the~#1~delimiter~of~your~
10864   \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10865   in~that~column.
10866   \bool_if:NT \l_@@_submatrix_slim_bool
10867   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10868   This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10869 }
10870 \@@_msg_new:nnn { width~without~X~columns }
10871 {
10872   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10873   the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10874   That~key~will~be~ignored.
10875 }
10876 {
10877   This~message~is~the~message~'width~without~X~columns'~
10878   of~the~module~'nicematrix'.~
10879   The~experimented~users~can~disable~that~message~with~
10880   \token_to_str:N \msg_redirect_name:nnn .\\
10881 }
10882
10883 \@@_msg_new:nn { key~multiplicity~with~dotted }
10884 {
10885   Incompatible~keys. \\
10886   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10887   in~a~'custom~line'.~They~are~incompatible. \\
10888   The~key~'multiplicity'~will~be~discarded.
10889 }
10890 \@@_msg_new:nn { empty~environment }
10891 {
10892   Empty~environment.\\
10893   Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10894 }
10895 \@@_msg_new:nn { No~letter~and~no~command }
10896 {
10897   Erroneous~use.\\
10898   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10899   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10900   ~'ccommand'~(to~draw~horizontal~rules).\\

```

```

10901     However,~you~can~go~on.
10902 }
10903 \@@_msg_new:nn { Forbidden~letter }
10904 {
10905     Forbidden~letter.\\
10906     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10907     It~will~be~ignored.\\
10908     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10909 }
10910 \@@_msg_new:nn { Several~letters }
10911 {
10912     Wrong~name.\\
10913     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10914     have~used~' \l_@@_letter_str ').\\
10915     It~will~be~ignored.
10916 }
10917 \@@_msg_new:nn { Delimiter~with~small }
10918 {
10919     Delimiter~forbidden.\\
10920     You~can't~put~a~delimiter~in~the~preamble~of~your~
10921     \@@_full_name_env: \
10922     because~the~key~'small'~is~in~force.\\
10923     This~error~is~fatal.
10924 }
10925 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10926 {
10927     Unknown~cell.\\
10928     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10929     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10930     can't~be~executed~because~a~cell~doesn't~exist.\\
10931     This~command~ \token_to_str:N \line \ will~be~ignored.
10932 }
10933 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10934 {
10935     Duplicate~name.\\
10936     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10937     in~this~ \@@_full_name_env: .\\
10938     This~key~will~be~ignored.\\
10939     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10940     { For~a~list~of~the~names~already~used,~type~H<return>. }
10941 }
10942 {
10943     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10944     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10945 }
10946 \@@_msg_new:nn { r~or~l~with~preamble }
10947 {
10948     Erroneous~use.\\
10949     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10950     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10951     your~ \@@_full_name_env: .\\
10952     This~key~will~be~ignored.
10953 }
10954 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10955 {
10956     Erroneous~use.\\
10957     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10958     in~an~exterior~column~of~
10959     the~array.~This~error~is~fatal.
10960 }

```

```

10961 \@@_msg_new:nn { bad-corner }
10962 {
10963   Bad-corner.\
10964   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10965   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
10966   This~specification~of~corner~will~be~ignored.
10967 }
10968 \@@_msg_new:nn { bad-border }
10969 {
10970   Bad-border.\
10971   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10972   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10973   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10974   also~use~the~key~'tikz'
10975   \IfPackageLoadedF { tikz }
10976   { ~if~you~load~the~LaTeX~package~'tikz' } ).\
10977   This~specification~of~border~will~be~ignored.
10978 }
10979 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10980 {
10981   TikZ~not~loaded.\
10982   You~can't~use~ \token_to_str:N \TikzEveryCell \
10983   because~you~have~not~loaded~TikZ.\
10984   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10985   before~or~after~'nicematrix'. \
10986   This~command~will~be~ignored.
10987 }
10988 \@@_msg_new:nn { tikz-key~without~tikz }
10989 {
10990   TikZ~not~loaded.\
10991   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10992   \Block '~because~you~have~not~loaded~TikZ.\
10993   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
10994   before~or~after~'nicematrix'. \
10995   This~key~will~be~ignored.
10996 }
10997 \@@_msg_new:nn { Bad~argument~for~Block }
10998 {
10999   Bad~argument.\
11000   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11001   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11002   '#1'. \
11003   If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
11004   the~argument~was~empty).
11005 }
11006 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
11007 {
11008   Erroneous~use.\
11009   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11010   'last-col'~without~value.\
11011   However,~you~can~go~on~for~this~time~
11012   (the~value~' \l_keys_value_tl '~will~be~ignored).
11013 }
11014 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
11015 {
11016   Erroneous~use. \
11017   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11018   'last-col'~without~value. \
11019   However,~you~can~go~on~for~this~time~
11020   (the~value~' \l_keys_value_tl '~will~be~ignored).
11021 }

```

```

11022 \@@_msg_new:nn { Block-too-large-1 }
11023 {
11024   Block-too-large. \\
11025   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is-
11026   too-small-for-that-block. \\
11027   This-block-and-maybe-others-will-be-ignored.
11028 }
11029 \@@_msg_new:nn { Block-too-large-2 }
11030 {
11031   Block-too-large. \\
11032   The-preamble-of-your- \@@_full_name_env: \ announces- \int_use:N
11033   \g_@@_static_num_of_col_int \
11034   columns-but-you-use-only- \int_use:N \c@jCol \ and-that's-why-a-block-
11035   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands-
11036   (&)~at-the-end-of-the-first-row-of-your- \@@_full_name_env: . \\
11037   This-block-and-maybe-others-will-be-ignored.
11038 }
11039 \@@_msg_new:nn { unknown-column-type }
11040 {
11041   Bad-column-type. \\
11042   The-column-type-#1'-in-your- \@@_full_name_env: \
11043   is-unknown. \\
11044   This-error-is-fatal.
11045 }
11046 \@@_msg_new:nn { unknown-column-type-multicolumn }
11047 {
11048   Bad-column-type. \\
11049   The-column-type-#1'-in-the-command-\token_to_str:N \multicolumn \
11050   of-your- \@@_full_name_env: \
11051   is-unknown. \\
11052   This-error-is-fatal.
11053 }
11054 \@@_msg_new:nn { unknown-column-type-S }
11055 {
11056   Bad-column-type. \\
11057   The-column-type-'S'-in-your- \@@_full_name_env: \ is-unknown. \\
11058   If-you-want-to-use-the-column-type-'S'-of-siunitx,~you-should-
11059   load-that-package. \\
11060   This-error-is-fatal.
11061 }
11062 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
11063 {
11064   Bad-column-type. \\
11065   The-column-type-'S'-in-the-command-\token_to_str:N \multicolumn \
11066   of-your- \@@_full_name_env: \ is-unknown. \\
11067   If-you-want-to-use-the-column-type-'S'-of-siunitx,~you-should-
11068   load-that-package. \\
11069   This-error-is-fatal.
11070 }
11071 \@@_msg_new:nn { tabularnote-forbidden }
11072 {
11073   Forbidden-command. \\
11074   You-can't-use-the-command- \token_to_str:N \tabularnote \
11075   ~here.~This-command-is-available-only-in-
11076   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in-
11077   the-argument-of-a-command-\token_to_str:N \caption \ included-
11078   in-an-environment-\{table\}. \\
11079   This-command-will-be-ignored.
11080 }
11081 \@@_msg_new:nn { borders-forbidden }
11082 {

```

```

11083   Forbidden-key.\\
11084   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
11085   because~the~option~'rounded-corners'~
11086   is~in~force~with~a~non-zero~value.\\
11087   This~key~will~be~ignored.
11088 }
11089 \@@_msg_new:nn { bottomrule~without~booktabs }
11090 {
11091   booktabs~not~loaded.\\
11092   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
11093   loaded~'booktabs'.~You~should~load~'booktabs',~before~or~
11094   after~'nicematrix'.\\
11095   This~key~will~be~ignored.
11096 }
11097 \@@_msg_new:nn { enumitem~not~loaded }
11098 {
11099   enumitem~not~loaded. \\
11100   You~can't~use~the~command~ \token_to_str:N \tabularnote \
11101   ~because~you~haven't~loaded~'enumitem'.~We~should~load~it~
11102   (before~or~after~'nicematrix').\\
11103   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
11104   ignored~in~the~document.
11105 }
11106 \@@_msg_new:nn { tikz~without~tikz }
11107 {
11108   TikZ~not~loaded. \\
11109   You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~
11110   loaded.~If~you~go~on,~that~key~will~be~ignored.
11111 }
11112 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
11113 {
11114   TikZ~not~loaded. \\
11115   You~have~used~the~key~'tikz'~in~the~definition~of~a~
11116   customized~line~(with~'custom~line')~but~TikZ~is~not~loaded.~
11117   You~can~go~on~but~you~will~have~another~error~if~you~actually~
11118   use~that~custom~line.
11119 }
11120 \@@_msg_new:nn { tikz~in~borders~without~tikz }
11121 {
11122   TikZ~not~loaded. \\
11123   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11124   command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11125   That~key~will~be~ignored.
11126 }
11127 \@@_msg_new:nn { color~in~custom~line~with~tikz }
11128 {
11129   Erroneous~use.\\
11130   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
11131   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
11132   The~key~'color'~will~be~discarded.
11133 }
11134 \@@_msg_new:nn { Wrong~last~row }
11135 {
11136   Wrong~number.\\
11137   You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
11138   \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11139   If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11140   last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11141   problem~by~using~'last~row'~without~value~(more~compilations~
11142   might~be~necessary).
11143 }

```

```

11144 \@@_msg_new:nn { Yet-in-env }
11145 {
11146   Nested-environments.\
11147   Environments-of~nicematrix~can't-be-nested.~However-you~
11148   can-insert,~for-example,~a~\{tabular\}~in-a~\{NiceTabular\}~
11149   or~a~\{NiceTabular\}~in-a~\{tabular\}.~You-can-also-compose~
11150   an-environment-of~nicematrix~in-a-box-of~LaTeX~and-insert~
11151   that~box~in-another-environment-of~nicematrix.\
11152   This-error-is-fatal.
11153 }
11154 \@@_msg_new:nn { Outside-math-mode }
11155 {
11156   Outside-math-mode.\
11157   The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
11158   (and-not-in~ \token_to_str:N \vcenter ).\
11159   This-error-is-fatal.
11160 }
11161 \@@_msg_new:nn { One-letter-allowed }
11162 {
11163   Bad-name.\
11164   The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
11165   you-have-used~' \l_keys_value_tl '~.\
11166   It-will-be-ignored.
11167 }
11168 \@@_msg_new:nn { TabularNote-in-CodeAfter }
11169 {
11170   Environment~\{TabularNote\}~forbidden.\
11171   You-must-use~\{TabularNote\}~at-the-end-of~your~\{NiceTabular\}~
11172   but~*before*~the~ \token_to_str:N \CodeAfter . \
11173   This-environment~\{TabularNote\}~will-be-ignored.
11174 }
11175 \@@_msg_new:nn { varwidth-not-loaded }
11176 {
11177   varwidth-not-loaded.\
11178   You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
11179   loaded.~You-should-load~'varwidth',~before-or-after~'nicematrix'. \
11180   Your-column-will-behave-like~'p'.
11181 }
11182 \@@_msg_new:nn { varwidth-not-loaded-in-X }
11183 {
11184   varwidth-not-loaded.\
11185   You-can't-use-the-key~'V'~in~your~column~'X'~
11186   because~'varwidth'~is-not-loaded.~You-should-load~'varwidth',~
11187   before-or-after~'nicematrix'. \
11188   It-will-be-ignored. \
11189 }
11190 \@@_msg_new:nnn { Unknown-key-for-a-rule }
11191 {
11192   Unknown-key.\
11193   Your-key~' \l_keys_key_str '~is-unknown-for-a-rule.\
11194   \c_@@_available_keys_str
11195 }
11196 {
11197   The-available-keys-are:~color,~multiplicity,~sep-color,~tikz~
11198   and~total-width~(meaningful-only-in-cunjunction-with~'tikz').
11199 }

```

If fact, there is also the key dotted but it won't be very useful since we provide `\hdottedline`, `\cdottedline` and the letter `:`.

```

11200 \@@_msg_new:nnn { Unknown-key-for-Block }
11201 {

```

```

11202     Unknown~key. \\
11203     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11204     \token_to_str:N \Block . \\
11205     It~will~be~ignored. \\
11206     \c_@@_available_keys_str
11207 }
11208 {
11209     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11210     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11211     opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11212     transparent~and~vlines.
11213 }
11214 \@@_msg_new:nnn { Unknown~key~for~Brace }
11215 {
11216     Unknown~key. \\
11217     The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11218     \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
11219     It~will~be~ignored. \\
11220     \c_@@_available_keys_str
11221 }
11222 {
11223     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11224     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11225     right~shorten)~and~yshift.
11226 }
11227 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11228 {
11229     Unknown~key. \\
11230     The~key~' \l_keys_key_str '~is~unknown. \\
11231     It~will~be~ignored. \\
11232     \c_@@_available_keys_str
11233 }
11234 {
11235     The~available~keys~are~(in~alphabetic~order):~
11236     delimiters/color,~
11237     rules~(with~the~subkeys~'color'~and~'width'),~
11238     sub~matrix~(several~subkeys)~
11239     and~xdots~(several~subkeys).~
11240     The~latter~is~for~the~command~ \token_to_str:N \line .
11241 }
11242 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11243 {
11244     Unknown~key. \\
11245     The~key~' \l_keys_key_str '~is~unknown. \\
11246     It~will~be~ignored. \\
11247     \c_@@_available_keys_str
11248 }
11249 {
11250     The~available~keys~are~(in~alphabetic~order):~
11251     create~cell~nodes,~
11252     delimiters/color~and~
11253     sub~matrix~(several~subkeys).
11254 }
11255 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11256 {
11257     Unknown~key. \\
11258     The~key~' \l_keys_key_str '~is~unknown. \\
11259     That~key~will~be~ignored. \\
11260     \c_@@_available_keys_str
11261 }
11262 {
11263     The~available~keys~are~(in~alphabetic~order):~

```

```

11264     'delimiters/color',~
11265     'extra-height',~
11266     'hlines',~
11267     'hvlines',~
11268     'left-xshift',~
11269     'name',~
11270     'right-xshift',~
11271     'rules'~(with~the~subkeys~'color'~and~'width'),~
11272     'slim',~
11273     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
11274     and~'right-xshift').\\
11275 }
11276 \\@@_msg_new:nnn { Unknown~key~for~notes }
11277 {
11278     Unknown~key.\\
11279     The~key~' \l_keys_key_str '~is~unknown.\\
11280     That~key~will~be~ignored. \\
11281     \c_@@_available_keys_str
11282 }
11283 {
11284     The~available~keys~are~(in~alphabetic~order):~
11285     bottomrule,~
11286     code~after,~
11287     code~before(+),~
11288     detect~duplicates,~
11289     enumitem~keys,~
11290     enumitem~keys~para,~
11291     para,~
11292     label~in~list,~
11293     label~in~tabular~and~
11294     style.
11295 }
11296 \\@@_msg_new:nnn { Unknown~key~for~RowStyle }
11297 {
11298     Unknown~key.\\
11299     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11300     \token_to_str:N \RowStyle . \\
11301     That~key~will~be~ignored. \\
11302     \c_@@_available_keys_str
11303 }
11304 {
11305     The~available~keys~are~(in~alphabetic~order):~
11306     bold,~
11307     cell~space~top~limit(+),~
11308     cell~space~bottom~limit(+),~
11309     cell~space~limits(+),~
11310     color,~
11311     fill~(alias:~rowcolor),~
11312     nb~rows,~
11313     opacity~and~
11314     rounded~corners.
11315 }
11316 \\@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11317 {
11318     Unknown~key.\\
11319     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11320     \token_to_str:N \NiceMatrixOptions . \\
11321     That~key~will~be~ignored. \\
11322     \c_@@_available_keys_str
11323 }
11324 {
11325     The~available~keys~are~(in~alphabetic~order):~
11326     &-in~blocks,~

```

```

11327 allow-duplicate-names,~
11328 ampersand-in-blocks,~
11329 caption-above,~
11330 cell-space-bottom-limit(+),~
11331 cell-space-limits(+),~
11332 cell-space-top-limit(+),~
11333 code-for-first-col(+),~
11334 code-for-first-row(+),~
11335 code-for-last-col(+),~
11336 code-for-last-row(+),~
11337 corners,~
11338 custom-key,~
11339 create-extra-nodes,~
11340 create-medium-nodes,~
11341 create-large-nodes,~
11342 custom-line,~
11343 delimiters~(several~subkeys),~
11344 end-of-row,~
11345 first-col,~
11346 first-row,~
11347 h(v)lines,~
11348 h(v)lines-except-borders,~
11349 last-col,~
11350 last-row,~
11351 left-margin,~
11352 light-syntax,~
11353 light-syntax-expanded,~
11354 matrix/columns-type,~
11355 no-cell-nodes,~
11356 notes~(several~subkeys),~
11357 nullify-dots,~
11358 pgf-node-code,~
11359 renew-dots,~
11360 renew-matrix,~
11361 respect-arraystretch,~
11362 rounded-corners,~
11363 right-margin,~
11364 rules~(with~the~subkeys~'color'~and~'width'),~
11365 small,~
11366 sub-matrix~(several~subkeys),~
11367 vlines,~
11368 xdots~(several~subkeys).
11369 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

11370 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
11371 {
11372   Unknown~key.\\
11373   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11374   \{NiceArray\}. \\
11375   That~key~will~be~ignored. \\
11376   \c_@@_available_keys_str
11377 }
11378 {
11379   The~available~keys~are~(in~alphabetic~order):~
11380   &~in~blocks,~
11381   ampersand-in-blocks,~
11382   b,~
11383   baseline,~
11384   c,~
11385   cell-space-bottom-limit,~
11386   cell-space-limits,~
11387   cell-space-top-limit,~

```

```

11388     code-after,~
11389     code-for-first-col(+),~
11390     code-for-first-row(+),~
11391     code-for-last-col(+),~
11392     code-for-last-row(+),~
11393     columns-width,~
11394     corners,~
11395     create-blocks-in-col,~
11396     create-extra-nodes,~
11397     create-medium-nodes,~
11398     create-large-nodes,~
11399     draw-trees-in-col,~
11400     extra-left-margin,~
11401     extra-right-margin,~
11402     first-col,~
11403     first-row,~
11404     h(v)lines,~
11405     h(v)lines-except-borders,~
11406     last-col,~
11407     last-row,~
11408     left-margin,~
11409     light-syntax,~
11410     light-syntax-expanded,~
11411     name,~
11412     no-cell-nodes,~
11413     nullify-dots,~
11414     pgf-node-code,~
11415     renew-dots,~
11416     respect-arraystretch,~
11417     right-margin,~
11418     rounded-corners,~
11419     rules~(with~the~subkeys~'color'~and~'width'),~
11420     small,~
11421     t,~
11422     vlines,~
11423     xdots/color,~
11424     xdots/shorten-start(+),~
11425     xdots/shorten-end(+),~
11426     xdots/shorten(+)-and~
11427     xdots/line-style.
11428 }

```

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`).

```

11429 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11430 {
11431   Unknown~key.\\
11432   The~key~' \l_keys_key_str '~is~unknown~for~the~
11433   \@@_full_name_env: . \\
11434   That~key~will~be~ignored. \\
11435   \c_@@_available_keys_str
11436 }
11437 {
11438   The~available~keys~are~(in~alphabetic~order):~
11439   &-in-blocks,~
11440   ampersand-in-blocks,~
11441   b,~
11442   baseline,~
11443   c,~
11444   cell-space-bottom-limit,~
11445   cell-space-limits,~
11446   cell-space-top-limit,~
11447   code-after,~

```

```

11448 code-for-first-col(+),~
11449 code-for-first-row(+),~
11450 code-for-last-col(+),~
11451 code-for-last-row(+),~
11452 columns-type,~
11453 columns-width,~
11454 corners,~
11455 create-blocks-in-col,~
11456 create-extra-nodes,~
11457 create-medium-nodes,~
11458 create-large-nodes,~
11459 draw-trees-in-col,~
11460 extra-left-margin,~
11461 extra-right-margin,~
11462 first-col,~
11463 first-row,~
11464 h(v)lines,~
11465 h(v)lines-except-borders,~
11466 l,~
11467 last-col,~
11468 last-row,~
11469 left-margin,~
11470 light-syntax,~
11471 light-syntax-expanded,~
11472 name,~
11473 no-cell-nodes,~
11474 nullify-dots,~
11475 pgf-node-code,~
11476 r,~
11477 renew-dots,~
11478 respect-arraystretch,~
11479 right-margin,~
11480 rounded-corners,~
11481 rules~(with~the~subkeys~'color'~and~'width'),~
11482 small,~
11483 t,~
11484 vlines,~
11485 xdots/color,~
11486 xdots/shorten-start(+),~
11487 xdots/shorten-end(+),~
11488 xdots/shorten(+)-and~
11489 xdots/line-style.
11490 }

11491 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11492 {
11493   Unknown~key.\\
11494   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11495   \{NiceTabular\}. \\
11496   That~key~will~be~ignored. \\
11497   \c_@@_available_keys_str
11498 }
11499 {
11500   The~available~keys~are~(in~alphabetic~order):~
11501   &~in~blocks,~
11502   ampersand~in~blocks,~
11503   b,~
11504   baseline,~
11505   c,~
11506   caption,~
11507   cell-space-bottom-limit,~
11508   cell-space-limits,~
11509   cell-space-top-limit,~
11510   code~after,~

```

```

11511 code-for-first-col(+),~
11512 code-for-first-row(+),~
11513 code-for-last-col(+),~
11514 code-for-last-row(+),~
11515 columns-width,~
11516 corners,~
11517 custom-line,~
11518 create-blocks-in-col,~
11519 create-extra-nodes,~
11520 create-medium-nodes,~
11521 create-large-nodes,~
11522 draw-trees-in-col,~
11523 extra-left-margin,~
11524 extra-right-margin,~
11525 first-col,~
11526 first-row,~
11527 h(v)lines,~
11528 h(v)lines-except-borders,~
11529 label,~
11530 last-col,~
11531 last-row,~
11532 left-margin,~
11533 light-syntax,~
11534 light-syntax-expanded,~
11535 name,~
11536 no-cell-nodes,~
11537 notes~(several~subkeys),~
11538 nullify-dots,~
11539 pgf-node-code,~
11540 renew-dots,~
11541 respect-arraystretch,~
11542 right-margin,~
11543 rounded-corners,~
11544 rules~(with~the~subkeys~'color'~and~'width'),~
11545 short-caption,~
11546 t,~
11547 tabularnote,~
11548 vlines,~
11549 xdots/color,~
11550 xdots/shorten-start(+),~
11551 xdots/shorten-end(+),~
11552 xdots/shorten(+)-and~
11553 xdots/line-style.
11554 }

11555 \@@_msg_new:nnn { Duplicate-name }
11556 {
11557   Duplicate-name.\
11558   The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
11559   the-same-environment-name-twice.~You-can-go-on,~but,~
11560   maybe,~you-will-have-incorrect-results-especially~
11561   if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
11562   message-again,~use-the-key~'allow-duplicate-names'~in~
11563   ' \token_to_str:N \NiceMatrixOptions '~\
11564   \bool_if:NF \g_@@_messages_for_Overleaf_bool
11565     { For-a-list-of-the-names-already-used,~type-H~<return>. }
11566 }
11567 {
11568   The-names-already-defined-in~this~document~are:~
11569   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11570 }

11571 \@@_msg_new:nn { caption-above~in~env }
11572 {
11573   The-key~'caption-above'~must-be-used-in~\token_to_str:N \NiceMatrixOptions.\

```

```

11574     That~key~will~be~ignored.
11575 }
11576 \@@_msg_new:nn { show-cell-names }
11577 {
11578     There~is~no~key~'show-cell-names'~in~nicematrix.\\
11579     You~should~use~the~command~\token_to_str:N \ShowCellNames\
11580     in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11581     \CodeAfter. \\
11582     That~key~will~be~ignored.
11583 }
11584 \@@_msg_new:nn { Option~auto~for~columns~width }
11585 {
11586     Erroneous~use.\\
11587     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
11588     That~key~will~be~ignored.
11589 }
11590 \@@_msg_new:nn { NiceTabularX~without~X }
11591 {
11592     NiceTabularX~without~X.\\
11593     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
11594     However,~you~can~go~on.
11595 }
11596 \@@_msg_new:nn { Preamble~forgotten }
11597 {
11598     Preamble~forgotten.\\
11599     You~have~probably~forgotten~the~preamble~of~your~
11600     \@@_full_name_env: . \\
11601     This~error~is~fatal.
11602 }
11603 \@@_msg_new:nn { Invalid~col~number }
11604 {
11605     Invalid~column~number.\\
11606     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11607     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.\\
11608     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11609 }
11610 \@@_msg_new:nn { Invalid~row~number }
11611 {
11612     Invalid~row~number.\\
11613     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11614     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.\\
11615     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11616 }
11617 \@@_define_com:NNN p ( )
11618 \@@_define_com:NNN b [ ]
11619 \@@_define_com:NNN v | |
11620 \@@_define_com:NNN V \! \!
11621 \@@_define_com:NNN B \{ \}

```

# Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	53
10	The environment <code>{NiceArrayWithDelims}</code>	57
11	Construction of the preamble of the array	62
12	The redefinition of <code>\multicolumn</code>	79
13	The environment <code>{NiceMatrix}</code> and its variants	97
	13.1 Definition of <code>{pNiceMatrix}</code>	97
	13.2 The key <code>renew-matrix</code>	98
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	98
15	After the construction of the array	99
16	We draw the dotted lines	106
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	123
18	User commands available in the new environments	129
19	The command <code>\line</code> accessible in <code>\CodeAfter</code>	135
20	The command <code>\RowStyle</code>	137
21	Colors of cells, rows and columns	140
22	The vertical and horizontal rules	152
23	The empty corners	174
24	The environment <code>{NiceMatrixBlock}</code>	177
25	The extra nodes	178
26	The blocks	182
27	Automatic arrays	209
28	The redefinition of the command <code>\dotfill</code>	211
29	The command <code>\diagbox</code>	211

30	The keyword <code>\CodeAfter</code>	212
31	The delimiters in the preamble	213
32	The command <code>\SubMatrix</code>	214
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	223
34	The commands <code>HBrace</code> et <code>VBrace</code>	226
35	The command <code>TikzEveryCell</code>	229
36	The key <code>draw-trees-in-col</code>	231
37	The key <code>create-blocks-in-col</code>	232
38	The command <code>\ShowCellNames</code>	233
39	We process the options at package loading	235
40	About the package underscore	236
41	Compatibility with <code>threeparttable</code>	237
42	Error messages of the package	237