

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

May 22, 2026

## Abstract

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

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}
8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your~LaTeX~release-is-too-old. \\
11    You~need-at-least-the~version~of~2025-06-01. \\
12    If~you~use~Overleaf,~you~need-at-least~"TeXLive~2025".\\
13    The~package~'nicematrix'~won't~be~loaded.
14   }
15 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
16 \IfFormatAtLeastTF { 2025-06-01 }
17   { }
18 { \msg_critical:nn { nicematrix } { latex-too-old } }
```

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

```
19 \RequirePackage { amsmath }
```

---

\*This document corresponds to the version 7.9a of `nicematrix`, at the date of 2026/05/22.

```
20 \RequirePackage{array}
```

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

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

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

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

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

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”.

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

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

```
55 \hook_gput_code:nnn { begindocument / end } { . }
56 { \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`.

```
57 \IfClassLoadedTF { revtex4-1 }
58 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
59 {
```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

104 \tl_new:N \l_@@_argspec_tl

105 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
106 \cs_generate_variant:Nn \str_set:Nn { N o }
107 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
108 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
109 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
110 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
111 \cs_generate_variant:Nn \dim_min:nn { v }
112 \cs_generate_variant:Nn \dim_max:nn { v }

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

```

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.

```

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

```

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.

```

125 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
126 {
127   \iow_now:Nn \@mainaux
128   {

```

```

129      \ExplSyntaxOn
130      \cs_if_free:NT \pgfsyspdfmark
131      { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
132      \ExplSyntaxOff
133    }
134    \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
135  }

```

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.

```

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

```

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.

```

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

```

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

```

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`.

```

171 \AtBeginDocument

```

```

172 {
173   \cs_set_protected:Npe \@@_everycr:
174   {
175     \IfPackageLoadedTF { colortbl } \CT@everycr \everycr
176     { \noalign { \@@_in_everycr: } }
177   }
178   \IfPackageLoadedTF { colortbl }
179   {
180     \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
181     \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
182     \cs_new_protected:Npn \@@_revert_colortbl:
183     {
184       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
185       {
186         \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
187         \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
188       }
189     }

```

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).

```

190     \cs_new_protected:Npn \@@_replace_columncolor:
191     {
192       \tl_replace_all:Nnn \g_@@_array_preamble_tl
193       \columncolor
194       \@@_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.

```

195     }
196   }
197   {
198     \cs_new_protected:Npn \@@_revert_colortbl: { }
199     \cs_new_protected:Npn \@@_replace_columncolor:
200     { \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.

```

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

```

Idem for `\CT@drs@`.

```

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

```

```

224     }
225 }

```

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.

```

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

```

236     \skip_horizontal:N \c_zero_dim
237 }

```

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`.

```

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

```

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.

```

242 \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`.

```

243 { \@@_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*.

```

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

```

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).

```

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

```

---

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```



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

```

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

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

309 \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.

```

310 \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).

```

311 \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.

```

312 \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.

```

313 \box_new:N \l_@@_the_array_box

```

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

```

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

```

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

```

316 \bool_new:N \l_@@_tabular_bool

```

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

```

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

```

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

```

321 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool

```

The following counter will count the environments {NiceMatrixBlock}.

```
322 \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.

```
323 \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).

```
324 \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.).

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

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

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

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

```
330 \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.

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

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

Idem for the mono-row blocks.

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

```
336 \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`.

```
337 \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.

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

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

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

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

```
343 \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).

```
344 \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).

```
345 \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).

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

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

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

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

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

The following boolean will be raised when the command `\rotate` is used with the key `-90`.

```
349 \bool_new:N \g_@@_rotate_minus_bool
```

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

```
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 it up).

```
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:nnTF \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_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
679 \seq_put_right:Ne \l_@@_notes_labels_seq
680 {
681 \tl_if_novalue:nTF { #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 { \pgfpoint scale { 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  $\cdot$ . 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 package `nicematrix` 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:nNtT { \c@jCol } = { 1 }
{ \int_compare:nNtT \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 \@@_tuning_key_small: % 2026/04/06
1306 \xglobal \colorlet { nicematrix-first-row } { . }
1307 \fi:
1308 \else:
1309 \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
1310 \fi:
1311 }
```

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

We will use a version a little more efficient.

```

1312 \cs_new_protected:Npn \@@_tuning_last_row:
1313 {
1314   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1315     \l_@@_code_for_last_row_tl
1316     \@@_tuning_key_small: % 2026/04/06
1317     \xglobal \colorlet { nicematrix-last-row } { . }
1318   \fi:
1319 }

```

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

```

1320 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1321 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1322 {
1323   \m@th
1324   $ % $

```

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

```

1325   \@@_tuning_key_small:
1326 }
1327 \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.

```

1328 \cs_new_protected:Npn \@@_begin_of_row:
1329 {
1330   \int_gincr:N \c@iRow
1331   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1332   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1333   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1334   \pgfpicture
1335   \pgfrememberpicturepositiononpagetrue
1336   \pgfcoordinate
1337     { \@@_env: - row - \int_use:N \c@iRow - base }
1338     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1339   \str_if_empty:NF \l_@@_name_str
1340     {
1341       \pgfnodealias
1342         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1343         { \@@_env: - row - \int_use:N \c@iRow - base }
1344     }
1345   \endpgfpicture
1346 }

```

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.

```

1347 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1348 {
1349   \if_int_compare:w \c@iRow = \c_zero_int
1350     \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
1351       \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
1352     \fi:
1353     \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
1354       \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
1355     \fi:
1356   \else:
1357     \if_int_compare:w \c@iRow = \c_one_int
1358       \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
1359         \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
1360       \fi:
1361     \fi:
1362   \fi:
1363 }

1364 \cs_new_protected:Npn \@@_rotate_cell_box:
1365 {
1366   \box_rotate:Nn \l_@@_cell_box
1367   { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
1368   \bool_if:NTF \g_@@_rotate_c_bool
1369   {
1370     \hbox_set:Nn \l_@@_cell_box
1371     {
1372       \IfFormatAtLeastTF { 2026-04-01 }
1373       { \vbox_center:n { \box_use:N \l_@@_cell_box } }
1374       {
1375         \m@th
1376         $ % $
1377         \vcenter { \box_use:N \l_@@_cell_box }
1378         $ % $
1379       }
1380     }
1381   }
1382   {
1383     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1384     {
1385       \vbox_set_top:Nn \l_@@_cell_box
1386       {
1387         \vbox_to_zero:n { }
1388         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1389         \box_use:N \l_@@_cell_box
1390       }
1391     }
1392   }
1393   \bool_gset_false:N \g_@@_rotate_bool
1394   \bool_gset_false:N \g_@@_rotate_c_bool
1395   \bool_gset_false:N \g_@@_rotate_minus_bool

```



1396 }

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.

```
1397 \cs_set_protected:Npn \@@_adjust_size_box:
1398 {
1399   \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
1400     \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
1401       \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
1402     \fi:
1403     \global \g_@@_blocks_wd_dim = \c_zero_dim
1404   \fi:
1405   \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
1406     \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
1407       \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
1408     \fi
1409     \global \g_@@_blocks_dp_dim = \c_zero_dim
1410   \fi:
1411   \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
1412     \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
1413       \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
1414     \fi:
1415     \global \g_@@_blocks_ht_dim = \c_zero_dim
1416   \fi:
1417 }
1418 \cs_new_protected:Npn \@@_cell_end:
1419 {
```

The following command is nullified in the tabulars.

```
1420   \@@_tuning_not_tabular_end:
1421   \hbox_set_end:
1422   \@@_cell_end_i:
1423 }
```

```
\cs_new_protected:Npn \@@_cell_end_i:
{
  \g_@@_cell_after_hook_tl
  \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
  \@@_adjust_size_box:
  \box_set_ht:Nn \l_@@_cell_box
```

```

    { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
\box_set_dp:Nn \l_@@_cell_box
    { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
\@@_update_max_cell_width:
\@@_update_for_first_and_last_row:
\bool_if:NTF \g_@@_empty_cell_bool
{ \box_use_drop:N \l_@@_cell_box }
{
    \bool_if:NTF \g_@@_not_empty_cell_bool
    { \@@_print_node_cell: }
    {
        \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
        { \@@_print_node_cell: }
        { \box_use_drop:N \l_@@_cell_box }
    }
}
\int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
{ \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
\bool_gset_false:N \g_@@_empty_cell_bool
\bool_gset_false:N \g_@@_not_empty_cell_bool
}

```

Here is a version slightly more efficient.

```

1424 \cs_new_protected:Npn \@@_cell_end_i:
1425 {

```

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.

```

1426     \g_@@_cell_after_hook_tl
1427     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1428     \@@_adjust_size_box:
1429     \box_set_ht:Nn \l_@@_cell_box
1430     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1431     \box_set_dp:Nn \l_@@_cell_box
1432     { \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”).

```

1433     \@@_update_max_cell_width:

```

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

```

1434     \@@_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.

```

1435 \bool_if:NTF \g_@@_empty_cell_bool
1436 { \box_use_drop:N \l_@@_cell_box }
1437 {
1438   \bool_if:NTF \g_@@_not_empty_cell_bool
1439   \@@_print_node_cell:
1440   {
1441     \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
1442     \@@_print_node_cell:
1443     \else:
1444       \box_use_drop:N \l_@@_cell_box
1445     \fi:
1446   }
1447 }
1448 \if_int_compare:w \c@jCol > \g_@@_col_total_int
1449 \global \g_@@_col_total_int = \c@jCol
1450 \fi:
1451 \global \let \g_@@_empty_cell_bool \c_false_bool
1452 \global \let \g_@@_not_empty_cell_bool \c_false_bool
1453 }

```

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:

```

1454 \cs_new_protected:Npn \@@_update_max_cell_width:
1455 {
1456   \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
1457   \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
1458   \fi:
1459 }

```

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`).

```

1460 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1461 {
1462   \@@_math_toggle:
1463   \hbox_set_end:
1464   \bool_if:NF \g_@@_rotate_bool
1465   {
1466     \hbox_set:Nn \l_@@_cell_box
1467     {
1468       \makebox [ \l_@@_col_width_dim ] [ s ]
1469       { \hbox_unpack_drop:N \l_@@_cell_box }
1470     }
1471   }
1472   \@@_cell_end_i:
1473 }

```

```

1474 \pgfset
1475 {
1476   nicematrix / cell-node /.style =
1477   {
1478     inner-sep = \c_zero_dim ,
1479     minimum~width = \c_zero_dim
1480   }
1481 }

```

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.

```

1482 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1483 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1484 {
1485   \use:c
1486   {
1487     __siunitx_table_align_
1488     \bool_if:NTF \l__siunitx_table_text_bool
1489     \l__siunitx_table_align_text_tl
1490     \l__siunitx_table_align_number_str
1491     :n
1492   }
1493   { #1 }
1494 }

```

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!

```

1495 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1496 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1497 {
1498   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1499   \hbox:n
1500   {
1501     \pgfsys@markposition
1502     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1503   }
1504   #1
1505   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1506   \hbox:n
1507   {
1508     \pgfsys@markposition
1509     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1510   }
1511 }

1512 \cs_new_protected:Npn \@@_print_node_cell:
1513 {
1514   \socket_use:nn { nicematrix / siunitx-wrap }
1515   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1516 }

```

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

```

1517 \cs_new_protected:Npn \@@_node_cell:
1518 {
1519   \pgfpicture
1520   \pgfsetbaseline \c_zero_dim
1521   \pgfrememberpicturepositiononpagetrue
1522   \pgfset { nicematrix / cell-node }
1523   \pgfnode
1524   { rectangle }
1525   { base }
1526   {

```

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).

```

1527   \sys_if_engine_xetex:T { \set@color }

```

```

1528     \box_use:N \l_@@_cell_box
1529   }
1530   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1531   { \l_@@_pgf_node_code_tl }
1532   \str_if_empty:NF \l_@@_name_str
1533   {
1534     \pgfnodealias
1535     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1536     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1537   }
1538   \endpgfpicture
1539 }

```

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).

```

1540 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1541 {
1542   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1543   { g_@@_ #2 _ lines _ tl }
1544   {
1545     \use:c { @@ _ draw _ #2 : nnn }
1546     { \int_use:N \c@iRow }
1547     { \int_use:N \c@jCol }
1548     { \exp_not:n { #3 } }
1549   }
1550 }

1551 \cs_new_protected:Npn \@@_array:n
1552 {
1553   \dim_set:Nn \col@sep
1554   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1555   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1556   { \def \@halignto { } }
1557   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1558   \@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)`.

```

1559   [ \str_if_eq:eeTF c \l_@@_baseline_tl c t ]
1560 }
1561 \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`.

```
1562 \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign
```

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

```
1563 \cs_new_protected:Npn \@@_create_row_node:
1564 {
1565   \int_compare:nNt \c@iRow > \g_@@_last_row_node_int
1566   {
1567     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1568     \@@_create_row_node_i:
1569   }
1570 }

1571 \cs_new_protected:Npn \@@_create_row_node_i:
1572 {
```

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

```
1573   \hbox
1574   {
1575     \bool_if:NT \l_@@_code_before_bool
1576     {
1577       \vtop
1578       {
1579         \skip_vertical:N 0.5\arrayrulewidth
1580         \pgfsys@markposition
1581         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1582         \skip_vertical:N -0.5\arrayrulewidth
1583       }
1584     }
1585     \pgfpicture
1586     \pgfrememberpicturepositiononpagetrue
1587     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1588     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1589     \str_if_empty:NF \l_@@_name_str
1590     {
1591       \pgfnodealias
1592       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1593       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1594     }
1595     \endpgfpicture
1596   }
1597 }
```

```
1598 \cs_new_protected:Npn \@@_in_everycr:
1599 {
1600   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1601   \tbl_update_cell_data_for_next_row:
1602   \int_gzero:N \c@jCol
1603   \bool_gset_false:N \g_@@_after_col_zero_bool
1604   \bool_if:NF \g_@@_row_of_col_done_bool
1605   {
1606     \@@_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).

```
1607   \clist_if_empty:NF \l_@@_hlines_clist
1608   {
1609     \str_if_eq:eeF \l_@@_hlines_clist { all }
1610     {
1611       \clist_if_in:NeT
1612       \l_@@_hlines_clist
```

```

1613         { \int_eval:n { \c@iRow + 1 } }
1614     }
1615 }

```

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.

```

1616         \int_compare:nNnT \c@iRow > { -1 }
1617     {
1618         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1619         { \hrule height \arrayrulewidth width \c_zero_dim }
1620     }
1621 }
1622 }
1623 }
1624 }

```

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

```

1625 \cs_set_protected:Npn \@@_renew_dots:
1626 {
1627     \cs_set_eq:NN \ldots \@@_Ldots:
1628     \cs_set_eq:NN \cdots \@@_Cdots:
1629     \cs_set_eq:NN \vdots \@@_Vdots:
1630     \cs_set_eq:NN \ddots \@@_Ddots:
1631     \cs_set_eq:NN \iddots \@@_Iddots:
1632     \cs_set_eq:NN \dots \@@_Ldots:
1633     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1634 }

```

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

```

1635 \AtBeginDocument
1636 {
1637     \IfPackageLoadedTF { booktabs }
1638     {
1639         \cs_new_protected:Npn \@@_patch_booktabs:
1640         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1641     }
1642     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1643 }

```

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`.

```

1644 \cs_new_protected:Npn \@@_some_initialization:
1645 {
1646     \@@_everycr:
1647     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1648     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1649     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim

```

<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}`.

```

1650 \dim_gzero:N \g_@@_dp_ante_last_row_dim
1651 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1652 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1653 }

```

`\@@_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`.

```

1654 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1655 {

```

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

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

```

1656 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1657 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1658 \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.

```

1659 \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.

```

1660 \fp_gzero:N \g_@@_total_X_weight_fp
1661 \bool_gset_false:N \g_@@_V_of_X_bool

1662 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1663 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1664 \@@_patch_booktabs:
1665 \box_clear_new:N \l_@@_cell_box
1666 \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}`).

```

1667 \bool_if:NT \l_@@_small_bool
1668 {
1669     \def \arraystretch { 0.47 }
1670     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1671 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1672 }

```

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!

```

1673 \bool_if:NT \g_@@_create_cell_nodes_bool
1674 {
1675     \tl_put_right:Nn \@@_begin_of_row:
1676     {
1677         \pgfsys@markposition

```



```

1678         { \@@_env: - row - \int_use:N \c@iRow - base }
1679     }
1680     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1681 }

```

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`.

```

1682 \def \ar@ialign
1683 {
1684     \tbl_init_cell_data_for_table:
1685     \@@_some_initialization:
1686     \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`.

```

1687     \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1688     \halign
1689 }

```

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).

```

1690 \cs_set_eq:NN \@@_old_ldots: \ldots
1691 \cs_set_eq:NN \@@_old_cdots: \cdots
1692 \cs_set_eq:NN \@@_old_vdots: \vdots
1693 \cs_set_eq:NN \@@_old_ddots: \ddots
1694 \cs_set_eq:NN \@@_old_iddots: \iddots
1695 \bool_if:NTF \l_@@_standard_cline_bool
1696 { \cs_set_eq:NN \cline \@@_standard_cline: }
1697 { \cs_set_eq:NN \cline \@@_cline: }
1698 \cs_set_eq:NN \Ldots \@@_Ldots:
1699 \cs_set_eq:NN \Cdots \@@_Cdots:
1700 \cs_set_eq:NN \Vdots \@@_Vdots:
1701 \cs_set_eq:NN \Ddots \@@_Ddots:
1702 \cs_set_eq:NN \Iddots \@@_Iddots:
1703 \cs_set_eq:NN \Hline \@@_Hline:
1704 \cs_set_eq:NN \Hspace \@@_Hspace:
1705 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1706 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1707 \cs_set_eq:NN \Block \@@_Block:
1708 \cs_set_eq:NN \rotate \@@_rotate:
1709 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1710 \cs_set_eq:NN \dotfill \@@_dotfill:
1711 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1712 \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1713 \cs_if_free:NT \CodeBefore { \cs_set_eq:NN \CodeBefore \@@_CodeBefore: }
1714 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1715 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1716 \cs_set_eq:NN \TopRule \@@_TopRule
1717 \cs_set_eq:NN \MidRule \@@_MidRule
1718 \cs_set_eq:NN \BottomRule \@@_BottomRule
1719 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1720 \cs_set_eq:NN \Hbrace \@@_Hbrace
1721 \cs_set_eq:NN \Vbrace \@@_Vbrace
1722 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1723 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1724 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1725 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1726 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1727 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular

```

```

1728 \int_if_zero:nTF \l_@@_first_row_int
1729 { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
1730 {
1731     \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1732     { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
1733 }
1734 \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:`.

```

1735 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1736 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1737 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1738 \@@_revert_colortbl:

```

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

```

1739 \tl_if_exist:NT \l_@@_note_in_caption_tl
1740 {
1741     \tl_if_empty:NF \l_@@_note_in_caption_tl
1742     {
1743         \int_gset:Nn \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1744         \int_gset:Nn \c@tabularnote \l_@@_note_in_caption_tl
1745     }
1746 }

```

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).

```

1747 \seq_gclear:N \g_@@_multicolumn_cells_seq
1748 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

When we compose a cell which belongs to a column which contains a instruction `\columncolor` (in the preamble of the environment), we add the number of that column in the following sequence (in order to recall that we have written the following instruction of the `\g_@@_pre_code_before_tl`).

```

1749 \seq_gclear_new:N \g_@@_columncolor_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1750 \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`).

```

1751 \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.

```

1752 \int_gzero:N \g_@@_col_total_int
1753 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1754 \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.

```

1755 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1756 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1757 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1758 \tl_gclear_new:N \g_@@_Ddots_lines_tl

```

```

1759 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1760 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1761 \tl_gclear:N \g_nicematrix_code_before_tl
1762 \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]`).

```

1763 \dim_zero_new:N \l_@@_left_delim_dim
1764 \dim_zero_new:N \l_@@_right_delim_dim
1765 \bool_if:NTF \g_@@_delims_bool
1766 {

```

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

```

1767 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1768 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1769 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1770 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1771 }
1772 {
1773 \dim_gset:Nn \l_@@_left_delim_dim
1774 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1775 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1776 }
1777 }

```

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.

```

1778 \cs_new_protected:Npn \@@_pre_array:
1779 {
1780 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1781 \int_gzero_new:N \c@iRow
1782 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1783 \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).

```

1784 \int_compare:nNnT \l_@@_last_row_int > \c_zero_int
1785 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1786 \int_compare:nNnT \l_@@_last_col_int > \c_zero_int
1787 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1788 \bool_if:NT \g_@@_aux_found_bool
1789 {
1790 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1791 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1792 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1793 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1794 }

```

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).

```

1795 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1796 {

```

```

1797     \bool_set_true:N \l_@@_last_row_without_value_bool
1798     \bool_if:NT \g_@@_aux_found_bool
1799     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1800   }
1801   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1802   {
1803     \bool_if:NT \g_@@_aux_found_bool
1804     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1805   }

```

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”.

```

1806   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1807   {
1808     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1809     {
1810       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1811       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1812       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1813       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1814     }
1815   }

1816   \seq_gclear:N \g_@@_cols_vlism_seq
1817   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1818   \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

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

```

1819   \@@_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).

```

1820   \hbox_set:Nw \l_@@_the_array_box
1821   \skip_horizontal:N \l_@@_left_margin_dim % \skip_horizontal:N ?
1822   \skip_horizontal:N \l_@@_extra_left_margin_dim
1823   \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).

```

1824   \m@th
1825   $ % $
1826   \bool_if:NTF \l_@@_light_syntax_bool
1827   { \use:c { @@-light-syntax } }
1828   { \use:c { @@-normal-syntax } }
1829 }

```

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

```

1830 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1831 {
1832   \tl_set:Nn \l_tmpa_tl { #1 }
1833   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1834   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1835   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1836   \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...

```
1837 \@@_pre_array:
1838 }
```

## 9 The `\CodeBefore`

```
1839 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
1840 \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).

```
1841 \cs_new_protected:Npn \@@_pre_code_before:
1842 {
```

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.

```
1843 \pgfsys@markposition { \@@_env: - position }
1844 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1845 \pgfpicture
1846 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1847 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1848 {
1849 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1850 \pgfcoordinate { \@@_env: - row - ##1 }
1851 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1852 }
```

Now, the recreation of the `col` nodes.

```
1853 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1854 {
1855 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1856 \pgfcoordinate { \@@_env: - col - ##1 }
1857 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1858 }
```

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

```
1859 \bool_if:NT \g_@@_create_cell_nodes_bool \@@_recreate_cell_nodes:
1860 \endpgfpicture
```

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

```
1861 \@@_create_diag_nodes:
```

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

```
1862 \@@_create_blocks_nodes:
1863 \IfPackageLoadedT { tikz }
1864 {
1865 \tikzset
1866 {
1867 every~picture / .style =
1868 { overlay , name~prefix = \@@_env: - }
1869 }
1870 }
1871 \cs_set_eq:NN \cellcolor \@@_cellcolor
1872 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1873 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1874 \cs_set_eq:NN \rowcolor \@@_rowcolor
1875 \cs_set_eq:NN \rowcolors \@@_rowcolors
```

```

1876 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1877 \cs_set_eq:NN \arraycolor \@@_arraycolor
1878 \cs_set_eq:NN \columncolor \@@_columncolor
1879 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1880 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1881 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1882 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNamesCodeBefore
1883 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1884 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1885 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1886 }

1887 \cs_new_protected:Npn \@@_exec_code_before:
1888 {

```

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...

```

1889 \clist_map_inline:Nn \l_@@_corners_cells_clist
1890 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1891 \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`.

```

1892 \@@_add_to_colors_seq:nn { { nocolor } } { }
1893 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1894 \group_begin:

```

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

```

1895 \if_mode_math:
1896 \@@_exec_code_before_i:
1897 \else:
1898 $ % $
1899 \@@_exec_code_before_i:
1900 $ % $
1901 \fi:
1902 \group_end:
1903 }

```

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`).

```

1904 \cs_new_protected:Npn \@@_exec_code_before_i:
1905 {
1906 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1907 { \@@_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`.

```

1908 \exp_last_unbraced:N \@@_CodeBefore_keys:
1909 \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.

```

1910 \@@_actually_color:
1911 \l_@@_code_before_tl
1912 \q_stop
1913 }

```

```

1914 \keys_define:nn { nicematrix / CodeBefore }
1915 {
1916   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1917   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1918   sub-matrix .value_required:n = true ,
1919   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1920   delimiters / color .value_required:n = true ,
1921   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1922 }
1923 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1924 {
1925   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1926   \@@_CodeBefore:w
1927 }

```

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.

```

1928 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1929 {
1930   \bool_if:NTF \g_@@_aux_found_bool
1931   {
1932     \@@_pre_code_before:
1933     \legacy_if:nF { measuring@ } { #1 }
1934   }

```

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).

```

1935   {
1936     \pgfsys@markposition { \@@_env: - position }
1937     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1938     \pgfpicture
1939     \pgf@relevantforpicturesizefalse
1940     \endpgfpicture

```

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

```

1941     \pgfpicture
1942     \pgfrememberpicturepositiononpagetrue
1943     \endpgfpicture

```

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

```

1944     \pgfpicture
1945     \pgf@relevantforpicturesizefalse
1946     \pgfrememberpicturepositiononpagetrue
1947     \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1948     \pgfpicture
1949     \pgf@relevantforpicturesizefalse
1950     \endpgfpicture
1951   }
1952 }

```

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.

```

1953 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1954 {
1955   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1956   {
1957     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1958     \pgfcoordinate { \@@_env: - row - ##1 - base }
1959     { \pgfpointdiff \@@_picture_position: \@@_node_position: }

```

```

1960 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1961 {
1962   \cs_if_exist:cT
1963   { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1964   {
1965     \pgfsys@getposition
1966     { \@@_env: - ##1 - #####1 - NW }
1967     \@@_node_position:
1968     \pgfsys@getposition
1969     { \@@_env: - ##1 - #####1 - SE }
1970     \@@_node_position_i:
1971     \@@_pgf_rect_node:nnn
1972     { \@@_env: - ##1 - #####1 }
1973     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1974     { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1975   }
1976 }
1977 }
1978 \@@_create_extra_nodes:
1979 \@@_create_aliases_last:
1980 }

1981 \cs_new_protected:Npn \@@_create_aliases_last:
1982 {
1983   \int_step_inline:nn \c@iRow
1984   {
1985     \pgfnodealias
1986     { \@@_env: - ##1 - last }
1987     { \@@_env: - ##1 - \int_use:N \c@jCol }
1988   }
1989   \int_step_inline:nn \c@jCol
1990   {
1991     \pgfnodealias
1992     { \@@_env: - last - ##1 }
1993     { \@@_env: - \int_use:N \c@iRow - ##1 }
1994   }
1995   \pgfnodealias
1996   { \@@_env: - last - last }
1997   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1998 }

```

```

1999 \cs_new_protected:Npn \@@_create_blocks_nodes:
2000 {
2001   \pgfpicture
2002   \pgf@relevantforpicturesizefalse
2003   \pgfrememberpicturepositiononpagetrue
2004   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
2005   { \@@_create_one_block_node:nnnnn ##1 }
2006   \endpgfpicture
2007 }

```

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>

```

2008 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
2009 {
2010   \tl_if_empty:nF { #5 }
2011   {
2012     \@@_qpoint:n { col - #2 }
2013     \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

---

<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 \@@_qpoint:n { #1 }
2015 \dim_set_eq:NN \l_tmpb_dim \pgf@y
2016 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
2017 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
2018 \@@_qpoint:n { \int_eval:n { #3 + 1 } }
2019 \@@_pgf_rect_node:nnnnn
2020 { \@@_env: - #5 }
2021 { \dim_use:N \l_tmpa_dim }
2022 { \dim_use:N \l_tmpb_dim }
2023 { \dim_use:N \l_@@_tmpc_dim }
2024 { \dim_use:N \pgf@y }
2025 }
2026 }

```

## 10 The environment {NiceArrayWithDelims}

```

2027 \NewDocumentEnvironment { NiceArrayWithDelims }
2028 { m m 0 { } m ! 0 { } t \CodeBefore }
2029 {
2030 \@@_provide_pgfsyspdfmark:
2031 \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.

```

2032 \bgroup

2033 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2034 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2035 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2036 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

2037 \int_gzero:N \g_@@_block_box_int
2038 \dim_gzero:N \g_@@_width_last_col_dim
2039 \dim_gzero:N \g_@@_width_first_col_dim
2040 \bool_gset_false:N \g_@@_row_of_col_done_bool
2041 \str_if_empty:NT \g_@@_name_env_str
2042 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2043 \bool_if:NTF \l_@@_tabular_bool
2044 \mode_leave_vertical:
2045 \@@_test_if_math_mode:
2046 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2047 \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.

```

2048 \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.

```

2049 \cs_if_exist:NT \tikz@library@external@loaded
2050 {
2051 \tikzexternaldisable
2052 \cs_if_exist:NT \ifstandalone
2053 { \tikzset { external / optimize = false } }

```

---

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

```
2054 }
```

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

```
2055 \int_gincr:N \g_@@_env_int
2056 \bool_if:NF \l_@@_block_auto_columns_width_bool
2057 { \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.

```
2058 \seq_gclear:N \g_@@_blocks_seq
2059 \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`.

```
2060 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2061 \seq_gclear:N \g_@@_pos_of_xdots_seq
2062 \tl_gclear_new:N \g_@@_code_before_tl
2063 \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.

```
2064 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2065 {
2066   \bool_gset_true:N \g_@@_aux_found_bool
2067   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2068 }
2069 { \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.

```
2070 \tl_gclear:N \g_@@_aux_tl
2071 \tl_if_empty:NF \g_@@_code_before_tl
2072 {
2073   \bool_set_true:N \l_@@_code_before_bool
2074   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2075 }
2076 \tl_if_empty:NF \g_@@_pre_code_before_tl
2077 { \bool_set_true:N \l_@@_code_before_bool }
```

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

```
2078 \bool_if:NTF \g_@@_delims_bool
2079 { \keys_set:nn { nicematrix / pNiceArray } }
2080 { \keys_set:nn { nicematrix / NiceArray } }
2081 { #3 , #5 }

2082 \@@_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:`.

```
2083 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
2084 }
```

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

```
2085 {
2086   \bool_if:NTF \l_@@_light_syntax_bool
2087   { \use:c { end @@-light-syntax } }
2088   { \use:c { end @@-normal-syntax } }
2089   $ % $
2090   \skip_horizontal:N \l_@@_right_margin_dim
```

```

2091 \skip_horizontal:N \l_@@_extra_right_margin_dim
2092 \hbox_set_end:
2093 \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.

```

2094 \bool_if:NT \l_@@_width_used_bool
2095 {
2096   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2097   { \@@_error_or_warning:n { width~without~X~columns } }
2098 }

```

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  $x$ .

```

2099 \fp_compare:nNnT \g_@@_total_X_weight_fp > \c_zero_fp
2100 \@@_compute_width_X:

```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```

2101 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2102 {
2103   \bool_if:NF \l_@@_last_row_without_value_bool
2104   {
2105     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
2106     {
2107       \@@_error:n { Wrong~last~row }
2108       \int_set_eq:NN \l_@@_last_row_int \c_iRow
2109     }
2110   }
2111 }

```

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>

```

2112 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2113 \bool_if:NTF \g_@@_last_col_found_bool
2114 { \int_gdecr:N \c@jCol }
2115 {
2116   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2117   { \@@_error:n { last~col~not~used } }
2118 }

```

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

```

2119 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2120 \int_compare:nNnT \l_@@_last_row_int > { -1 }
2121 { \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).

```

2122 \int_if_zero:nT \l_@@_first_col_int
2123 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2124 \bool_if:nTF { ! \g_@@_delims_bool }
2125 {
2126   \str_if_eq:eeTF c \l_@@_baseline_tl
2127   \@@_use_arraybox_with_notes_c:
2128   {

```

---

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

```

2129         \str_if_eq:eeTF b \l_@@_baseline_tl
2130         \@@_use_arraybox_with_notes_b:
2131         \@@_use_arraybox_with_notes:
2132     }
2133 }

```

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).

```

2134 {
2135     \int_if_zero:nTF \l_@@_first_row_int
2136     {
2137         \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2138         \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2139     }
2140     { \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>

```

2141     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2142     {
2143         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2144         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2145     }
2146     { \dim_zero:N \l_tmpb_dim }
2147     \hbox_set:Nn \l_tmpa_box
2148     {
2149         \m@th
2150         $ % $
2151         \@@_color:o \l_@@_delimiters_color_tl
2152         \exp_after:wN \left \g_@@_left_delim_tl
2153         \vcenter
2154         {

```

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.

```

2155         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2156         \hbox
2157         {
2158             \bool_if:NTF \l_@@_tabular_bool
2159             { \skip_horizontal:n { - \tabcolsep } }
2160             { \skip_horizontal:n { - \arraycolsep } }
2161             \@@_use_arraybox_with_notes_c:
2162             \bool_if:NTF \l_@@_tabular_bool
2163             { \skip_horizontal:n { - \tabcolsep } }
2164             { \skip_horizontal:n { - \arraycolsep } }
2165         }

```

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

```

2166         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2167     }
2168     \exp_after:wN \right \g_@@_right_delim_tl
2169     $ % $
2170 }

```

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.

```

2171     \bool_if:NTF \l_@@_delimiters_max_width_bool
2172     { \@@_put_box_in_flow_bis:nn \g_@@_left_delim_tl \g_@@_right_delim_tl }
2173     \@@_put_box_in_flow:
2174 }

```

---

<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).

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).

```

2175 \bool_if:NT \g_@@_last_col_found_bool
2176 { \skip_horizontal:N \g_@@_width_last_col_dim }
2177 \bool_if:NT \l_@@_preamble_bool
2178 {
2179   \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2180   { \@@_err_columns_not_used: }
2181 }
2182 \@@_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.

```

2183 \egroup

```

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

```

2184 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2185 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2186 \iow_now:Ne \@mainaux
2187 {
2188   \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2189   \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2190   { \exp_not:o \g_@@_aux_tl }
2191 }
2192 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2193 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2194 }

```

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

```

2195 \cs_new_protected:Npn \@@_err_columns_not_used:
2196 {
2197   \@@_warning:n { columns~not~used }
2198   \cs_gset:Npn \@@_err_columns_not_used: { }
2199 }

```

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  $x$ .

```

2200 \cs_new_protected:Npn \@@_compute_width_X:
2201 {
2202   \tl_gput_right:Ne \g_@@_aux_tl
2203   {
2204     \bool_set_true:N \l_@@_X_columns_aux_bool
2205     \dim_set:Nn \l_@@_X_columns_dim
2206     {

```

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`

```

2207   \bool_lazy_and:nnTF \g_@@_V_of_X_bool \l_@@_X_columns_aux_bool
2208   { \dim_use:N \l_@@_X_columns_dim }
2209   {
2210     \dim_compare:nNnTF
2211     {
2212       \dim_abs:n
2213       { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2214     }
2215     <
2216     { 0.001 pt }
2217     { \dim_use:N \l_@@_X_columns_dim }

```

```

2218         {
2219             \dim_eval:n
2220             {
2221                 \l_@@_X_columns_dim
2222                 +
2223                 \fp_to_dim:n
2224                 {
2225                     (
2226                         \dim_eval:n
2227                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2228                     )
2229                     / \fp_use:N \g_@@_total_X_weight_fp
2230                 }
2231             }
2232         }
2233     }
2234 }
2235 }
2236 }

```

## 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`.

```

2237 \cs_new_protected:Npn \@@_transform_preamble:
2238 {
2239     \@@_transform_preamble_i:
2240     \@@_transform_preamble_ii:
2241 }
2242 \cs_new_protected:Npn \@@_transform_preamble_i:
2243 {
2244     \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 `vlsim`).

```

2245     \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.

```

2246     \bool_gset_false:N \g_tmpb_bool

```

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

```

2247     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2248     \int_zero:N \l_tmpa_int
2249     \tl_gclear:N \g_@@_array_preamble_tl
2250     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2251     {
2252         \tl_gset:Nn \g_@@_array_preamble_tl
2253         { ! { \skip_horizontal:N \arrayrulewidth } }
2254     }
2255     {
2256         \clist_if_in:NnT \l_@@_vlines_clist 1
2257         {
2258             \tl_gset:Nn \g_@@_array_preamble_tl
2259             { ! { \skip_horizontal:N \arrayrulewidth } }
2260         }
2261     }

```

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

```

2262 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2263 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2264 \@@_replace_columncolor:
2265 }

2266 \cs_new_protected:Npn \@@_transform_preamble_ii:
2267 {

```

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

```

2268 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2269 {
2270 \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2271 { \bool_gset_true:N \g_@@_delims_bool }
2272 }
2273 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2274 \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).

```

2275 \int_if_zero:nTF \l_@@_first_col_int
2276 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2277 {
2278 \bool_if:NF \g_@@_delims_bool
2279 {
2280 \bool_if:NF \l_@@_tabular_bool
2281 {
2282 \clist_if_empty:NT \l_@@_vlines_clist
2283 {
2284 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2285 { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2286 }
2287 }
2288 }
2289 }
2290 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2291 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2292 {
2293 \bool_if:NF \g_@@_delims_bool
2294 {
2295 \bool_if:NF \l_@@_tabular_bool
2296 {
2297 \clist_if_empty:NT \l_@@_vlines_clist
2298 {
2299 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2300 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2301 }
2302 }
2303 }
2304 }

```

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.

```

2305 \tag_if_active:F
2306 {

```

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

```

2307     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2308     {
2309         \tl_gput_right:Nn \g_@@_array_preamble_tl
2310         { > { \@@_err_too_many_cols: } 1 }
2311     }
2312 }
2313 }

```

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.

```

2314 \cs_new_protected:Npn \@@_rec_preamble:n #1
2315 {

```

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>

```

2316     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2317     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2318     {

```

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

```

2319     \cs_if_exist:cTF { NC @ find @ #1 }
2320     {
2321         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2322         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2323     }
2324     {
2325         \str_if_eq:nnTF { #1 } { S }
2326         { \@@_fatal:n { unknown~column~type~S } }
2327         { \@@_fatal:nn { unknown~column~type } { #1 } }
2328     }
2329 }
2330 }

```

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

```

2331 \cs_new_protected:Npn \@@_c: #1
2332 {
2333     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2334     \tl_gclear:N \g_@@_pre_cell_tl
2335     \tl_gput_right:Nn \g_@@_array_preamble_tl
2336     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2337     \int_gincr:N \c@jCol
2338     \@@_rec_preamble_after_col:n
2339 }

2340 \cs_new_protected:Npn \@@_l: #1
2341 {
2342     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2343     \tl_gclear:N \g_@@_pre_cell_tl
2344     \tl_gput_right:Nn \g_@@_array_preamble_tl
2345     {

```

---

<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`.



```

2346         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2347         l
2348         < \@@_cell_end:
2349     }
2350     \int_gincr:N \c@jCol
2351     \@@_rec_preamble_after_col:n
2352 }
2353 \cs_new_protected:Npn \@@_r: #1
2354 {
2355     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2356     \tl_gclear:N \g_@@_pre_cell_tl
2357     \tl_gput_right:Nn \g_@@_array_preamble_tl
2358     {
2359         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2360         r
2361         < \@@_cell_end:
2362     }
2363     \int_gincr:N \c@jCol
2364     \@@_rec_preamble_after_col:n
2365 }

```

For ! and @

```

2366 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2367 {
2368     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2369     \@@_rec_preamble:n
2370 }
2371 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2372 \cs_new_protected:cpn { @@ _ | : } #1
2373 {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

2374     \int_incr:N \l_tmpa_int
2375     \@@_make_preamble_i_i:n
2376 }
2377 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2378 {

```

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

```

2379     \str_if_eq:nnTF { #1 } { | }
2380     { \use:c { @@ _ | : } | }
2381     { \@@_make_preamble_i_ii:nn { } #1 }
2382 }

```

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

```

2383 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2384 {
2385     \str_if_eq:nnTF { #2 } { [ ]
2386     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2387     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2388 }
2389 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2390 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2391 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2392 {
2393     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2394     \tl_gput_right:Ne \g_@@_array_preamble_tl
2395     {

```

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

```

2396     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_before_dim }
2397   }
2398   \tl_gput_right:Nx \g_@@_rules_tl
2399   {

```

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:

```

2400   {
2401     \int_compare:nNnT \l_tmpa_int > 1
2402     { \@@_set_multiplicity:n { \int_use:N \l_tmpa_int } }
2403     \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
2404     \dim_set:Nn \l_@@_rule_width_after_dim
2405     { \dim_use:N \l_@@_rule_width_before_dim }
2406     \@@_draw_vrule:n { #2 }
2407   }

```

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.

```

2408   }
2409   \int_zero:N \l_tmpa_int
2410   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2411   \@@_rec_preamble:n #1
2412 }

2413 \cs_new_protected:cpn { @@_ > : } #1 #2
2414 {
2415   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2416   \@@_rec_preamble:n
2417 }

2418 \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.

```

2419 \keys_define:nn { nicematrix / p-column }
2420 {
2421   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2422   r .value_forbidden:n = true ,
2423   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2424   c .value_forbidden:n = true ,
2425   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2426   l .value_forbidden:n = true ,
2427   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2428   S .value_forbidden:n = true ,
2429   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2430   p .value_forbidden:n = true ,
2431   t .meta:n = p ,
2432   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2433   m .value_forbidden:n = true ,
2434   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2435   b .value_forbidden:n = true
2436 }

```

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

```

2437 \cs_new_protected:Npn \@@_p: #1
2438 {
2439   \str_set:Nn \l_@@_vpos_col_str { #1 }
Now, you look for a potential character [ after the letter of the specifier (for the options).
2440   \@@_make_preamble_ii_i:n
2441 }
2442 \cs_new_eq:NN \@@_b: \@@_p:
2443 \cs_new_eq:NN \@@_m: \@@_p:
2444 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2445 {
2446   \str_if_eq:nnTF { #1 } { [ ]
2447     { \@@_make_preamble_ii_ii:w [ ]
2448       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2449     }
2450 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2451 { \@@_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.

```

2452 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2453 {

```

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).

```

2454   \str_set:Nn \l_@@_hpos_col_str { j }
2455   \@@_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`.

```

2456   \setlength { \l_tmpa_dim } { #2 }
2457   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2458 }
2459 \cs_new_protected:Npn \@@_keys_p_column:n #1
2460 { \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.

```

2461 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2462 {

```

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

```

2463   \use:e
2464   {
2465     \@@_make_preamble_ii_vi:nnnnnnnn
2466     { \str_if_eq:eeTF p \l_@@_vpos_col_str { t } { b } }
2467     { #1 }
2468     {
2469       \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.

```

2470       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2471       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2472       {

```

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

```

2473         \def \exp_not:N \l_@@_hpos_cell_tl
2474         { \str_lowercase:f { \l_@@_hpos_col_str } }
2475     }
2476     \IfPackageLoadedTF { ragged2e }
2477     {
2478         \str_case:on \l_@@_hpos_col_str
2479         {

```

The following `\exp_not:N` are mandatory.

```

2480         c { \exp_not:N \Centering }
2481         l { \exp_not:N \RaggedRight }
2482         r { \exp_not:N \RaggedLeft }
2483     }
2484 }
2485 {
2486     \str_case:on \l_@@_hpos_col_str
2487     {
2488         c { \exp_not:N \centering }
2489         l { \exp_not:N \raggedright }
2490         r { \exp_not:N \raggedleft }
2491     }
2492 }
2493 #3
2494 }
2495 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2496 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2497 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2498 { #2 }
2499 {
2500     \str_case:onF \l_@@_hpos_col_str
2501     {
2502         { j } { c }
2503         { si } { c }
2504     }

```

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

```

2505         { \str_lowercase:f \l_@@_hpos_col_str }
2506     }
2507 }

```

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

```

2508     \int_gincr:N \c@jCol
2509     \@@_rec_preamble_after_col:n
2510 }

```

**#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*.

```

2511 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2512 {
2513     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2514     {

```

```

2515     \tl_gput_right:Nn \g_@@_array_preamble_tl
2516     { > \@@_test_if_empty_for_S: }
2517   }
2518   {
2519     \str_if_eq:eeTF { #7 } { varwidth }
2520     {
2521       \tl_gput_right:Nn \g_@@_array_preamble_tl
2522       { > \@@_test_if_empty_varwidth: }
2523     }
2524     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2525   }
2526   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2527   \tl_gclear:N \g_@@_pre_cell_tl
2528   \tl_gput_right:Nn \g_@@_array_preamble_tl
2529   {
2530     > {

```

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.

```

2531     \dim_set:Nn \l_@@_col_width_dim { #2 }
2532     \@@_cell_begin:

```

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

```

2533     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2534     \everypar
2535     {
2536       \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2537       \everypar { }
2538     }

```

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

```

2539     #3

```

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

```

2540     \g_@@_row_style_tl
2541     \arraybackslash
2542     #5
2543   }
2544   #8
2545   < {
2546     #6

```

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

```

2547     \@finalstrut \@arstrutbox
2548     \use:c { end #7 }

```

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

```

2549     #4
2550     \@@_cell_end:
2551   }
2552 }
2553 }

```

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

```

2554 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2555 {

```

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 `&`).

```

2556 \group_align_safe_begin:
2557 \peek_meaning:NTF &
2558 \@@_the_cell_is_empty:
2559 {
2560   \peek_meaning:NTF \\\
2561   \@@_the_cell_is_empty:
2562   {
2563     \peek_meaning:NTF \crcr
2564     \@@_the_cell_is_empty:
2565     \group_align_safe_end:
2566   }
2567 }
2568 }

```

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

```

2569 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2570 {
2571   \group_align_safe_begin:
2572   \peek_meaning:NTF &
2573   \@@_the_cell_is_empty_varwidth:
2574   {
2575     \peek_meaning:NTF \\\
2576     \@@_the_cell_is_empty_varwidth:
2577     {
2578       \peek_meaning:NTF \crcr
2579       \@@_the_cell_is_empty_varwidth:
2580       \group_align_safe_end:
2581     }
2582   }
2583 }
2584 \cs_new_protected:Npn \@@_the_cell_is_empty:
2585 {
2586   \group_align_safe_end:
2587   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2588   {

```

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

```

2589   \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).

```

2590   \skip_horizontal:N \l_@@_col_width_dim
2591 }
2592 }
2593 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2594 {
2595   \group_align_safe_end:
2596   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2597   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2598 }
2599 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2600 {
2601   \peek_meaning:NT \__siunitx_table_skip:n
2602   { \bool_gset_true:N \g_@@_empty_cell_bool }
2603 }

```

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.

```
2604 \cs_new_protected:Npn \l_@@_center_cell_box:
2605 {
```

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

```
2606 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2607 {
2608   \dim_compare:nNnT
2609     { \box_ht:N \l_@@_cell_box }
2610     >
```

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).

```
2611   { \box_ht:N \strutbox }
2612   {
2613     \hbox_set:Nn \l_@@_cell_box
2614     {
2615       \box_move_down:nn
2616       {
2617         ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2618           + \baselineskip ) / 2
2619       }
2620       { \box_use:N \l_@@_cell_box }
2621     }
2622   }
2623 }
2624 }
```

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

```
2625 \cs_new_protected:Npn \l_@@_V: #1 #2
2626 {
2627   \str_if_eq:nnTF { #2 } { [ ] }
2628     { \@@_make_preamble_V_i:w [ ] }
2629     { \@@_make_preamble_V_i:w [ ] { #2 } }
2630 }
2631 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2632 { \@@_make_preamble_V_ii:nn { #1 } }
2633 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2634 {
2635   \str_set:Nn \l_@@_vpos_col_str { p }
2636   \str_set:Nn \l_@@_hpos_col_str { j }
2637   \@@_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`.

```
2638   \setlength { \l_tmpa_dim } { #2 }
2639   \IfPackageLoadedTF { varwidth }
2640     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2641     {
2642       \@@_error_or_warning:n { varwidth-not-loaded }
2643       \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2644     }
2645 }
2646 % \end{macrocode}
2647 %
2648 % \medskip
2649 % For |w| and |W|
2650 % \begin{macrocode}
2651 \cs_new_protected:Npn \l_@@_w: { \@@_make_preamble_w:nnnn { } }
```

```

2652 \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.

2653 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3
2654 {
2655   \tl_if_in:nnTF { clr } { #3 }
2656   { \@@_make_preamble_w_i:nnnn { #1 } { #2 } { #3 } }
2657   {
2658     \@@_error:nn { Invalid-argument-for~w } { #3 }
2659     \@@_make_preamble_w_i:nnnn { #1 } { #2 } { c }
2660   }
2661 }

2662 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2 #3 #4
2663 {
2664   \str_if_eq:nnTF { #3 } { s }
2665   { \@@_make_preamble_w_ii:nnnn { #1 } { #4 } }
2666   { \@@_make_preamble_w_iii:nnnn { #1 } { #2 } { #3 } { #4 } }
2667 }

```

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.

```

2668 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2
2669 {
2670   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2671   \tl_gclear:N \g_@@_pre_cell_tl
2672   \tl_gput_right:Nn \g_@@_array_preamble_tl
2673   {
2674     > {

```

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.

```

2675       \setlength { \l_@@_col_width_dim } { #2 }
2676       \@@_cell_begin:
2677       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2678     }
2679     c
2680   < {
2681     \@@_cell_end_for_w_s:
2682     #1
2683     \@@_adjust_size_box:
2684     \box_use_drop:N \l_@@_cell_box
2685   }
2686 }
2687 \int_gincr:N \c@jCol
2688 \@@_rec_preamble_after_col:n
2689 }

```

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

```

2690 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2691 {
2692   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2693   \tl_gclear:N \g_@@_pre_cell_tl
2694   \tl_gput_right:Nn \g_@@_array_preamble_tl
2695   {
2696     > {

```



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`.

```

2697         \setlength { \l_@@_col_width_dim } { #4 }
2698         \hbox_set:Nw \l_@@_cell_box
2699         \@@_cell_begin:
2700         \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2701     }
2702     c
2703     < {
2704         \@@_cell_end:
2705         \hbox_set_end:
2706         #1
2707         \@@_adjust_size_box:
2708         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2709     }
2710 }
```

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

```

2711     \int_gincr:N \c@jCol
2712     \@@_rec_preamble_after_col:n
2713 }

2714 \cs_new_protected:Npn \@@_special_W:
2715 {
2716     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2717     { \@@_warning:n { W~warning } }
2718 }
```

For `S` (of `siunitx`).

```

2719 \AtBeginDocument
2720 {
2721     \IfPackageLoadedT { siunitx }
2722     {
2723         \cs_new_protected:Npn \@@_S: #1 #2
2724         {
2725             \str_if_eq:nnTF { #2 } { [ ] }
2726             { \@@_make_preamble_S:w [ ] }
2727             { \@@_make_preamble_S:w [ ] { #2 } }
2728         }
2729     }
2730 }

2731 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2732 { \@@_make_preamble_S:i:n { #1 } }

2733 \cs_new_protected:Npn \@@_test_siunitx:
2734 {
2735     \IfPackageAtLeastF { siunitx } { 2026/03/26 }
2736     { \@@_fatal:n { siunitx~too-old } }
2737     \cs_gset_eq:NN \@@_test_siunitx: \prg_do_nothing:
2738 }
2739 % \end{macrocode}
2740 %
2741 % \begin{macrocode}
2742 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2743 {
2744     \@@_test_siunitx:
2745     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2746     \tl_gclear:N \g_@@_pre_cell_tl
2747     \tl_gput_right:Nn \g_@@_array_preamble_tl
2748     {
2749         > {
```

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).

```

2750         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2751         \keys_set:nn { siunitx } { #1 }
2752         \@@_cell_begin:
2753         \siunitx_cell_begin:w
2754     }
2755     c
2756     <
2757     {
2758         \siunitx_cell_end:

```

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

```

2759         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2760         {
2761             \bool_if:NTF \l__siunitx_table_text_bool
2762                 \bool_set_true:N
2763                 \bool_set_false:N
2764             \l__siunitx_table_text_bool
2765         }
2766         \@@_cell_end:
2767     }
2768 }

```

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

```

2769     \int_gincr:N \c@jCol
2770     \@@_rec_preamble_after_col:n
2771 }

```

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

```

2772 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2773 {
2774     \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.

```

2775     \int_if_zero:nTF \c@jCol
2776     {
2777         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2778         {

```

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

```

2779         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2780         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2781         \@@_rec_preamble:n #2
2782     }
2783     {
2784         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2785         \@@_make_preamble_iv:nn { #1 } { #2 }
2786     }
2787 }
2788 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2789 }
2790 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2791 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2792 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2793 {
2794     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2795     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2796 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2797 {
2798   \@@_error:nn { delimiter~after~opening } { #2 }
2799   \@@_rec_preamble:n
2800 }
2801 { \@@_rec_preamble:n #2 }
2802 }

```

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

```

2803 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2804 { \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}`).

```

2805 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2806 {
2807   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2808   \tl_if_in:nnTF { ) ] \} } { #2 }
2809   { \@@_make_preamble_v:nnn #1 #2 }
2810   {
2811     \str_if_eq:nnTF \s_stop { #2 }
2812     {
2813       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2814       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2815       {
2816         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2817         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2818         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2819         \@@_rec_preamble:n #2
2820       }
2821     }
2822     {
2823       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2824       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2825       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2826       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2827       \@@_rec_preamble:n #2
2828     }
2829   }
2830 }
2831 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2832 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2833 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2834 {
2835   \str_if_eq:nnTF \s_stop { #3 }
2836   {
2837     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2838     {
2839       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2840       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2841       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2842       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2843     }
2844     {
2845       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2846       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2847       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2848       \@@_error:nn { double~closing~delimiter } { #2 }
2849     }
2850   }
2851   {

```

```

2852      \tl_gput_right:Nn \g_@@_pre_code_after_tl
2853      { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2854      \@@_error:nn { double~closing~delimiter } { #2 }
2855      \@@_rec_preamble:n #3
2856    }
2857  }

2858 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2859 { \use:c { @@ _ \token_to_str:N ) : } }

```

After a specifier of column, we have to test whether there is one or several  $\langle \{ \dots \}$  because, after those potential  $\langle \{ \dots \}$ , we have to insert  $!\{\text{\skip\_horizontal:N} \dots\}$  when the key `vlines` is used. In fact, we have also to test whether there is, after the  $\langle \{ \dots \}$ , a  $\@ \{ \dots \}$ .

```

2860 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2861 {
2862   \str_if_eq:nnTF { #1 } { < }
2863   { \@@_rec_preamble_after_col_i:n }
2864   {
2865     \str_if_eq:nnTF { #1 } { @ }
2866     { \@@_rec_preamble_after_col_ii:n }
2867     {
2868       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2869       {
2870         \tl_gput_right:Nn \g_@@_array_preamble_tl
2871         { ! { \skip_horizontal:N \arrayrulewidth } }
2872       }
2873       {
2874         \clist_if_in:NiT \l_@@_vlines_clist
2875         { \int_eval:n { \c@jCol + 1 } }
2876         {
2877           \tl_gput_right:Nn \g_@@_array_preamble_tl
2878           { ! { \skip_horizontal:N \arrayrulewidth } }
2879         }
2880       }
2881       \@@_rec_preamble:n { #1 }
2882     }
2883   }
2884 }

2885 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2886 {
2887   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2888   \@@_rec_preamble_after_col:n
2889 }

```

We have to catch a  $\@ \{ \dots \}$  after a specifier of column because, if we have to draw a vertical rule, we have to add in that  $\@ \{ \dots \}$  a `\hskip` corresponding to the width of the vertical rule.

```

2890 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2891 {
2892   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2893   {
2894     \tl_gput_right:Nn \g_@@_array_preamble_tl
2895     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2896   }
2897   {
2898     \clist_if_in:NiT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2899     {
2900       \tl_gput_right:Nn \g_@@_array_preamble_tl
2901       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2902     }
2903     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2904   }
2905   \@@_rec_preamble:n
2906 }

```

```

2907 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2908 {
2909   \tl_clear:N \l_tmpa_tl
2910   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2911   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2912 }

```

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.

```

2913 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2914 { \@@_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`.

```

2915 \cs_new_protected:Npn \@@_X: #1 #2
2916 {
2917   \str_if_eq:nnTF { #2 } { [ ]
2918     { \@@_make_preamble_X:w [ ] }
2919     { \@@_make_preamble_X:w [ ] #2 }
2920   }
2921   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2922   { \@@_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`.

```

2923 \keys_define:nn { nicematrix / X-column }
2924 {
2925   V .code:n =
2926     \IfPackageLoadedTF { varwidth }
2927     {
2928       \bool_set_true:N \l_@@_V_of_X_bool
2929       \bool_gset_true:N \g_@@_V_of_X_bool
2930     }
2931     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2932   unknown .code:n =
2933     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2934     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2935     { \@@_error_or_warning:n { invalid~weight } }
2936 }

```

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

```

2937 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2938 {

```

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`).

```

2939   \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`).

```

2940   \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`).

```

2941   \fp_set:Nn \l_tmpa_fp { 1.0 }
2942   \@@_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.

```
2943 \bool_set_false:N \l_@@_V_of_X_bool
2944 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

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

```
2945 \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).

```
2946 \bool_if:NTF \l_@@_X_columns_aux_bool
2947 {
2948   \@@_make_preamble_ii_iv:nnn
```

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

```
2949 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2950 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2951 { \@@_no_update_width: }
2952 }
```

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.

```
2953 {
2954   \tl_gput_right:Nn \g_@@_array_preamble_tl
2955   {
2956     > {
2957       \@@_cell_begin:
2958       \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`.

```
2959 \NotEmpty
```

The following code will nullify the box of the cell.

```
2960 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2961 { \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`.

```
2962 \begin { minipage } { 5 cm } \arraybackslash
2963 }
2964 c
2965 < {
2966   \end { minipage }
2967   \@@_cell_end:
2968 }
2969 }
2970 \int_gincr:N \c@jCol
2971 \@@_rec_preamble_after_col:n
2972 }
2973 }
```

```
2974 \cs_new_protected:Npn \@@_no_update_width:
2975 {
2976   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2977   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2978 }
```

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

```

2979 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2980 {
2981   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2982   { \int_eval:n { \c@jCol + 1 } }
2983   \tl_gput_right:Ne \g_@@_array_preamble_tl
2984   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2985   \@@_rec_preamble:n
2986 }

```

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.

```

2987 \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).

```

2988 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2989 { \@@_fatal:n { Preamble-forgotten } }
2990 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2991 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2992 { @@ _ \token_to_str:N \hline : }
2993 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2994 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2995 { @@ _ \token_to_str:N \hline : }
2996 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2997 { @@ _ \token_to_str:N \hline : }
2998 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2999 { @@ _ \token_to_str:N \hline : }
3000 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
3001 { @@ _ \token_to_str:N \hline : }
3002 \cs_new_protected:cpn { @@ _ \token_to_str:N \linewidth : }
3003 { \@@_fatal:n { NiceTabularX~probably~required } }
3004 \cs_set_eq:cc { @@ _ \token_to_str:N \textwidth : }
3005 { @@ _ \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).

```

3006 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
3007 {

```

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`.

```

3008   \multispan { #1 }
3009   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
3010   \begingroup
3011   \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

3012   \tl_gclear:N \g_@@_preamble_tl
3013   \@@_make_m_preamble:n #2 \q_stop

```

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

```

3014   \def \@addamp
3015   {
3016     \legacy_if:nTF { @firstamp }
3017     { \legacy_if_set_false:n { @firstamp } }

```

```

3018         { \@preamerr 5 }
3019     }
3020     \exp_args:No \@mkpream \g_@@_preamble_tl
3021     \@addtopreamble \@empty
3022     \endgroup
3023     \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

3024     \int_compare:nNtT { #1 } > 1
3025     {
3026         \seq_gput_right:Ne \g_@@_multicolumn_cells_seq
3027         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
3028         \seq_gput_right:Nn \g_@@_multicolumn_sizes_seq { #1 }
3029         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
3030         {
3031             {
3032                 \int_if_zero:nTF \c@jCol
3033                 { \int_eval:n { \c@iRow + 1 } }
3034                 { \int_use:N \c@iRow }
3035             }
3036             { \int_eval:n { \c@jCol + 1 } }
3037             {
3038                 \int_if_zero:nTF \c@jCol
3039                 { \int_eval:n { \c@iRow + 1 } }
3040                 { \int_use:N \c@iRow }
3041             }
3042             { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

3043         { }
3044     }
3045 }

```

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

```

3046     \RenewDocumentCommand { \cellcolor } { 0 { } m }
3047     {
3048         \tl_gput_right:Ne \g_@@_pre_code_before_tl
3049         {
3050             \@_rectanglecolor [ ##1 ]
3051             { \exp_not:n { ##2 } }
3052             { \int_use:N \c@iRow - \int_use:N \c@jCol }
3053             { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
3054         }
3055         \ignorespaces
3056     }

```

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

```

3057     \def \@sharp { #3 }
3058     \@arstrut
3059     \@preamble
3060     \null

```

We add some lines.

```

3061     \int_gadd:Nn \c@jCol { #1 - 1 }
3062     \int_compare:nNtT \c@jCol > \g_@@_col_total_int
3063     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3064     \ignorespaces
3065 }

```



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`.

```

3066 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3067 {
3068   \str_case:nnF { #1 }
3069   {
3070     c { \@@_make_m_preamble_i:n #1 }
3071     l { \@@_make_m_preamble_i:n #1 }
3072     r { \@@_make_m_preamble_i:n #1 }
3073     > { \@@_make_m_preamble_ii:nn #1 }
3074     ! { \@@_make_m_preamble_ii:nn #1 }
3075     @ { \@@_make_m_preamble_ii:nn #1 }
3076     | { \@@_make_m_preamble_iii:n #1 }
3077     p { \@@_make_m_preamble_iv:nnn t #1 }
3078     m { \@@_make_m_preamble_iv:nnn c #1 }
3079     b { \@@_make_m_preamble_iv:nnn b #1 }
3080     w { \@@_make_m_preamble_v:nnnn { } #1 }
3081     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3082     \q_stop { }
3083   }
3084   {
3085     \cs_if_exist:cTF { NC @ find @ #1 }
3086     {
3087       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3088       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3089     }
3090     {
3091       \str_if_eq:nnTF { #1 } { S }
3092       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3093       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3094     }
3095   }
3096 }

```

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

```

3097 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3098 {
3099   \tl_gput_right:Nn \g_@@_preamble_tl
3100   {
3101     > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3102     #1
3103     < \@@_cell_end:
3104   }

```

We test for the presence of a `<`.

```

3105   \@@_make_m_preamble_x:n
3106 }

```

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

```

3107 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3108 {
3109   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3110   \@@_make_m_preamble:n
3111 }

```

For `|`

```

3112 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3113 {
3114   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3115   \@@_make_m_preamble:n
3116 }

```

For p, m and b

```

3117 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3118 {
3119   \tl_gput_right:Nn \g_@@_preamble_tl
3120   {
3121     > {
3122       \@@_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`.

```

3123       \setlength { \l_tmpa_dim } { #3 }
3124       \begin { minipage } [ #1 ] { \l_tmpa_dim }
3125       \mode_leave_vertical:
3126       \arraybackslash
3127       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
3128     }
3129     c
3130     < {
3131       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
3132       \end { minipage }
3133       \@@_cell_end:
3134     }
3135   }

```

We test for the presence of a `<`.

```

3136   \@@_make_m_preamble_x:n
3137 }

```

For w and W

```

3138 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3139 {
3140   \tl_gput_right:Nn \g_@@_preamble_tl
3141   {
3142     > {
3143       \dim_set:Nn \l_@@_col_width_dim { #4 }
3144       \hbox_set:Nw \l_@@_cell_box
3145       \@@_cell_begin:
3146       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3147     }
3148     c
3149     < {
3150       \@@_cell_end:
3151       \hbox_set_end:
3152       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3153       #1
3154       \@@_adjust_size_box:
3155       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3156     }
3157   }

```

We test for the presence of a `<`.

```

3158   \@@_make_m_preamble_x:n
3159 }

```

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

```

3160 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3161 {
3162   \str_if_eq:nnTF { #1 } { < }
3163   \@@_make_m_preamble_ix:n
3164   { \@@_make_m_preamble:n { #1 } }
3165 }

```

```

3166 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3167 {
3168   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3169   \@@_make_m_preamble_x:n
3170 }

```

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`).

```

3171 \cs_new_protected:Npn \@@_put_box_in_flow:
3172 {
3173   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3174   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3175   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3176   { \box_use_drop:N \l_tmpa_box }
3177   \@@_put_box_in_flow_i:
3178 }

```

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

```

3179 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3180 {
3181   \pgfpicture
3182   \@@_qpoint:n { row - 1 }
3183   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3184   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3185   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3186   \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).

```

3187   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3188   {
3189     \int_set:Nn \l_tmpa_int
3190     { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3191     \bool_lazy_or:nnT
3192     { \int_compare_p:nNn \l_tmpa_int < { 1 } }
3193     { \int_compare_p:nNn \l_tmpa_int > { \c@iRow + 1 } }
3194     {
3195       \@@_error:n { bad-value-for~baseline-line }
3196       \int_set:Nn \l_tmpa_int 1
3197     }
3198     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3199   }
3200   {
3201     \str_if_eq:eeTF t \l_@@_baseline_tl
3202     { \int_set:Nn \l_tmpa_int 1 }
3203     {
3204       \str_if_eq:eeTF b \l_@@_baseline_tl
3205       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3206       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3207     }
3208     \bool_lazy_or:nnT
3209     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3210     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3211     {
3212       \@@_error:n { bad-value-for~baseline }
3213       \int_set:Nn \l_tmpa_int 1
3214     }
3215     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3216         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3217     }
3218     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3219 \endpgfpicture
3220 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3221 % \box_use_drop:N \l_tmpa_box % 2026/04/13
3222 }

```

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).

```

3223 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3224 {

```

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.

```

3225     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3226     {
3227         \int_compare:nNnT \c@jCol > 1
3228         {
3229             \box_set_wd:Nn \l_@@_the_array_box
3230             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3231         }
3232     }

```

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

```

3233 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3234 \bool_if:NT \l_@@_caption_above_bool
3235 {
3236     \tl_if_empty:NF \l_@@_caption_tl
3237     {
3238         \bool_set_false:N \g_@@_caption_finished_bool
3239         \int_gzero:N \c@tabularnote
3240         \@@_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).

```

3241         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3242         {
3243             \tl_gput_right:Ne \g_@@_aux_tl
3244             {
3245                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3246                 { \int_use:N \g_@@_notes_caption_int }
3247             }
3248             \int_gzero:N \g_@@_notes_caption_int
3249         }
3250     }
3251 }

```

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

```

3252 \hbox
3253 {
3254     \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.

```

3255     \@@_create_extra_nodes:
3256     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3257 }

```

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).

```

3258     \bool_lazy_any:nT
3259     {
3260         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3261         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3262         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3263     }
3264     {
3265         \bool_if:NTF \l_@@_notes_no_print_bool
3266         { \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes: }
3267         \@@_tabular_notes:
3268     }
3269     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3270     \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3271     \end { minipage }
3272 }

```

```

3273 \cs_new_protected:Npn \@@_insert_caption:
3274 {
3275     \tl_if_empty:NF \l_@@_caption_tl
3276     {
3277         \cs_if_exist:NTF \@capttype
3278         { \@@_insert_caption_i: }
3279         { \@@_error:n { caption~outside~float } }
3280     }
3281 }

```

```

3282 \cs_new_protected:Npn \@@_insert_caption_i:
3283 {
3284     \group_begin:

```

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

```

3285     \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.

```

3286     \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3287     \tl_if_empty:NTF \l_@@_short_caption_tl
3288     \caption
3289     { \caption [ \l_@@_short_caption_tl ] }
3290     { \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.

```

3291     \bool_if:NF \g_@@_caption_finished_bool
3292     {
3293         \bool_gset_true:N \g_@@_caption_finished_bool

```

```

3294     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3295     \int_gzero:N \c@tabularnote
3296   }
3297   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3298   \group_end:
3299 }
3300 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3301 {
3302   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3303   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3304 }
3305 \cs_set_protected:Npn \@@_tabular_notes_error:
3306 { \@@_error:n { Bad-use~of~NiceTabularNotes } }
3307 \cs_set_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3308 \cs_set_protected:Npn \@@_tabular_notes:
3309 {
3310   \cs_gset_eq:NN \NiceTabularNotes \@@_tabular_notes_error:
3311   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3312   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3313   \skip_vertical:N 0.65ex

```

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

```

3314   \group_begin:
3315   \l_@@_notes_code_before_tl
3316   \tl_if_empty:NF \g_@@_tabularnote_tl
3317   {
3318     \g_@@_tabularnote_tl \par
3319     \tl_gclear:N \g_@@_tabularnote_tl
3320   }

```

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.

```

3321   \int_compare:nNnT \c@tabularnote > \c_zero_int
3322   {
3323     \bool_if:NTF \l_@@_notes_para_bool
3324     {
3325       \begin { tabularnotes* }
3326       \seq_map_inline:Nn \g_@@_notes_seq
3327       { \@@_one_tabularnote:nn ##1 }
3328       \strut
3329       \end { tabularnotes* }

```

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

```

3330     \par
3331   }
3332   {
3333     \tabularnotes
3334     \seq_map_inline:Nn \g_@@_notes_seq
3335     { \@@_one_tabularnote:nn ##1 }
3336     \strut
3337     \endtabularnotes
3338   }
3339 }
3340 \unskip
3341 \group_end:
3342 \bool_if:NT \l_@@_notes_bottomrule_bool
3343 {
3344   \IfPackageLoadedTF { booktabs }
3345   {

```

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

```

3346     \skip_vertical:N \aboverulesep

```

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

```

3347         { \CT@arc@ \hrule height \heavyrulewidth }
3348     }
3349     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3350 }
3351 \l_@@_notes_code_after_tl
3352 \seq_gclear:N \g_@@_notes_seq
3353 \seq_gclear:N \g_@@_notes_in_caption_seq
3354 \int_gzero:N \c@tabularnote
3355 }

```

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.

```

3356 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3357 {
3358     \tl_if_novalue:nTF { #1 }
3359     { \item }
3360     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3361 }

```

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.

```

3362 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3363 {
3364     \pgfpicture
3365     \@@_qpoint:n { row - 1 }
3366     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3367     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3368     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3369     \endpgfpicture
3370     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3371     \int_if_zero:nT \l_@@_first_row_int
3372     {
3373         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3374         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3375     }
3376     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3377 }

```

Now, the general case.

```

3378 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3379 {

```

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

```

3380     \str_if_eq:eeT \l_@@_baseline_tl { t }
3381     { \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`.

```

3382     \pgfpicture
3383     \@@_qpoint:n { row - 1 }
3384     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3385     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3386     {
3387         \int_set:Nn \l_tmpa_int
3388         {
3389             \str_range:Nnn
3390             \l_@@_baseline_tl
3391             { 6 }
3392             { \tl_count:o \l_@@_baseline_tl }

```

```

3393     }
3394     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3395   }
3396   {
3397     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3398     \bool_lazy_or:nnT
3399       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3400       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3401     {
3402       \@@_error:n { bad-value-for-baseline }
3403       \int_set:Nn \l_tmpa_int 1
3404     }
3405     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3406   }
3407   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3408   \endpgfpicture
3409   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3410   \int_if_zero:nT \l_@@_first_row_int
3411   {
3412     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3413     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3414   }
3415   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3416 }

```

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.

```

3417 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3418 {

```

We will compute the real width of both delimiters used.

```

3419   \dim_zero_new:N \l_@@_real_left_delim_dim
3420   \dim_zero_new:N \l_@@_real_right_delim_dim
3421   \hbox_set:Nn \l_tmpb_box
3422   {
3423     \m@th
3424     $ % $
3425     \left #1
3426     \vcenter
3427     {
3428       \vbox_to_ht:nn
3429         { \box_ht_plus_dp:N \l_tmpa_box }
3430       { }
3431     }
3432     \right .
3433     $ % $
3434   }
3435   \dim_set:Nn \l_@@_real_left_delim_dim
3436   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3437   \hbox_set:Nn \l_tmpb_box
3438   {
3439     \m@th
3440     $ % $
3441     \left .
3442     \vbox_to_ht:nn
3443       { \box_ht_plus_dp:N \l_tmpa_box }
3444     { }
3445     \right #2
3446     $ % $
3447   }
3448   \dim_set:Nn \l_@@_real_right_delim_dim
3449   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```



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

```

3450 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3451 \@@_put_box_in_flow:
3452 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3453 }

```

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).

```

3454 \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).

```

3455 {
3456   \peek_remove_spaces:n
3457   {
3458     \peek_meaning:NTF \end
3459     \@@_analyze_end:Nn
3460     {
3461       \@@_transform_preamble:
3462       \@@_array:o \g_@@_array_preamble_tl
3463     }
3464   }
3465 }
3466 {
3467   \@@_create_col_nodes:
3468   \endarray
3469 }

```

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

```

3470 \NewDocumentEnvironment { @@-light-syntax } { b }
3471 {

```

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`.

```

3472 \tl_if_empty:nT { #1 }
3473 { \@@_fatal:n { empty-environment } }
3474 \tl_if_in:nnT { #1 } { & }
3475 { \@@_fatal:n { ampersand-in-light-syntax } }
3476 \tl_if_in:nnT { #1 } { \ }
3477 { \@@_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`.

```

3478 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3479 }

```

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.

```

3480 {
3481   \@@_create_col_nodes:
3482   \endarray
3483 }

```

```

3484 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3485 {
3486   \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).

```

3487   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3488   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3489   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3490     \seq_set_split:Nee
3491     \seq_set_split:Non
3492     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3493   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3494   \tl_if_empty:NF \l_tmpa_tl
3495     { \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.

```

3496   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3497     { \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`).

```

3498   \tl_build_begin:N \l_@@_new_body_tl
3499   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3500   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3501   \@@_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).

```

3502   \seq_map_inline:Nn \l_@@_rows_seq
3503   {
3504     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3505     \@@_line_with_light_syntax:n { #1 }
3506   }
3507   \tl_build_end:N \l_@@_new_body_tl
3508   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3509   {
3510     \int_set:Nn \l_@@_last_col_int
3511     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3512   }

```

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.

```

3513   \@@_transform_preamble:

```

```

3514   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3515 }

```

```

3516 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3517 {
3518   \seq_clear_new:N \l_@@_cells_seq
3519   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3520   \int_set:Nn \l_@@_nb_cols_int
3521   { \int_max:nn \l_@@_nb_cols_int { \seq_count:N \l_@@_cells_seq } }
3522   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3523   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3524   \seq_map_inline:Nn \l_@@_cells_seq

```

```

3525     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3526   }
3527   \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`.

```

3528   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3529   {
3530     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3531     { \@@_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.

```

3532     \end { #2 }
3533   }

```

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`).

```

3534   \cs_new:Npn \@@_create_col_nodes:
3535   {
3536     \crrc
3537     \int_if_zero:nT \l_@@_first_col_int
3538     {
3539       \omit
3540       \hbox_overlap_left:n
3541       {
3542         \bool_if:NT \l_@@_code_before_bool
3543         { \pgfsys@markposition { \@@_env: - col - 0 } }
3544         \pgfpicture
3545         \pgfrememberpicturepositiononpagetrue
3546         \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3547         \str_if_empty:NF \l_@@_name_str
3548         { \pgfnodelalias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3549         \endpgfpicture
3550         \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3551       }
3552     }
3553   }
3554   \omit

```

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

```

3555     \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`).

```

3556     \int_if_zero:nTF \l_@@_first_col_int
3557     {
3558       \@@_mark_position:n { 1 }
3559       \pgfpicture
3560       \pgfrememberpicturepositiononpagetrue
3561       \pgfcoordinate { \@@_env: - col - 1 }
3562       { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3563       \str_if_empty:NF \l_@@_name_str
3564       { \pgfnodelalias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3565       \endpgfpicture
3566     }
3567     {
3568       \bool_if:NT \l_@@_code_before_bool
3569       {
3570         \hbox
3571         {

```

```

3572         \skip_horizontal:n { 0.5 \arrayrulewidth }
3573         \pgfsys@markposition { \@@_env: - col - 1 }
3574         \skip_horizontal:n { -0.5 \arrayrulewidth }
3575     }
3576 }
3577 \pgfpicture
3578 \pgfrememberpicturepositiononpagetrue
3579 \pgfcoordinate { \@@_env: - col - 1 }
3580 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3581 \@@_node_alias:n { 1 }
3582 \endpgfpicture
3583 }

```

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.

```

3584 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3585 \bool_if:NF \l_@@_auto_columns_width_bool
3586 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3587 {
3588     \bool_lazy_and:nnTF
3589     \l_@@_auto_columns_width_bool
3590     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3591     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3592     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3593     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3594 }
3595 \skip_horizontal:N \g_tmpa_skip
3596 \hbox
3597 {
3598     \@@_mark_position:n { 2 }
3599     \pgfpicture
3600     \pgfrememberpicturepositiononpagetrue
3601     \pgfcoordinate { \@@_env: - col - 2 }
3602     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3603     \@@_node_alias:n { 2 }
3604     \endpgfpicture
3605 }

```

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.

```

3606 \int_gset:Nn \g_tmpa_int 1
3607 \bool_if:NTF \g_@@_last_col_found_bool
3608 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3609 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3610 {
3611     &
3612     \omit
3613     \int_gincr:N \g_tmpa_int

```

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

```

3614     \skip_horizontal:N \g_tmpa_skip
3615     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3616 \pgfpicture
3617 \pgfrememberpicturepositiononpagetrue
3618 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3619 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3620 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3621 \endpgfpicture
3622 }

```

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).

```

3623 \bool_lazy_or:nnF
3624 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3625 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3626 {
3627   &
3628   \omit
3629   \skip_horizontal:N \g_tmpa_skip
3630   \int_gincr:N \g_tmpa_int
3631   \bool_lazy_any:nF
3632   {
3633     \g_@@_delims_bool
3634     \l_@@_tabular_bool
3635     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3636     \l_@@_exterior_arraycolsep_bool
3637     \l_@@_bar_at_end_of_pream_bool
3638   }
3639   { \skip_horizontal:n { - \col@sep } }
3640   \bool_if:NT \l_@@_code_before_bool
3641   {
3642     \hbox
3643     {
3644       \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.

```

3645 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3646 { \skip_horizontal:n { - \arraycolsep } }
3647 \pgfsys@markposition
3648 { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3649 \skip_horizontal:n { 0.5 \arrayrulewidth }
3650 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3651 { \skip_horizontal:N \arraycolsep }
3652 }
3653 }
3654 \pgfpicture
3655 \pgfrememberpicturepositiononpagetrue
3656 \pgfcoordinate { @@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3657 {
3658   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3659   {
3660     \pgfpoint
3661     { - 0.5 \arrayrulewidth - \arraycolsep }
3662     \c_zero_dim
3663   }
3664   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3665 }
3666 @@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3667 \endpgfpicture
3668 }

3669 \bool_if:NT \g_@@_last_col_found_bool
3670 {
3671   \hbox_overlap_right:n
3672   {
3673     \skip_horizontal:N \g_@@_width_last_col_dim
3674     \skip_horizontal:N \col@sep
3675     \bool_if:NT \l_@@_code_before_bool
3676     {
3677       \pgfsys@markposition
3678       { @@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }

```

```

3679         }
3680         \pgfpicture
3681         \pgfrememberpicturepositiononpagetrue
3682         \pgfcoordinate
3683         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3684         \pgfpointorigin
3685         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3686         \endpgfpicture
3687     }
3688 }
3689 }

3690 \cs_new_protected:Npn \@@_mark_position:n #1
3691 {
3692     \bool_if:NT \l_@@_code_before_bool
3693     {
3694         \hbox
3695         {
3696             \skip_horizontal:n { -0.5 \arrayrulewidth }
3697             \pgfsys@markposition { \@@_env: - col - #1 }
3698             \skip_horizontal:n { 0.5 \arrayrulewidth }
3699         }
3700     }
3701 }

3702 \cs_new_protected:Npn \@@_node_alias:n #1
3703 {
3704     \str_if_empty:NF \l_@@_name_str
3705     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3706 }

```

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

```

3707 \tl_const:Nn \c_@@_preamble_first_col_tl
3708 {
3709     >
3710     {

```

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

```

3711         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3712         \bool_gset_true:N \g_@@_after_col_zero_bool
3713         \@@_begin_of_row:
3714         \hbox_set:Nw \l_@@_cell_box
3715         \@@_math_toggle:
3716         \@@_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”.

```

3717         \int_compare:nNnT \c@iRow > \c_zero_int
3718         {
3719             \bool_lazy_or:nnT
3720             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3721             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3722             {
3723                 \l_@@_code_for_first_col_tl
3724                 \xglobal \colorlet { nicematrix-first-col } { . }
3725             }
3726         }
3727     }

```

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`.

```

3728     l
3729     <
3730     {
3731         \@@_math_toggle:
3732         \hbox_set_end:
3733         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3734         \@@_adjust_size_box:
3735         \@@_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.

```

3736         \dim_gset:Nn \g_@@_width_first_col_dim
3737         { \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.

```

3738         \hbox_overlap_left:n
3739         {
3740             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3741             \@@_node_cell:
3742             { \box_use_drop:N \l_@@_cell_box }
3743             \skip_horizontal:N \l_@@_left_delim_dim
3744             \skip_horizontal:N \l_@@_left_margin_dim
3745             \skip_horizontal:N \l_@@_extra_left_margin_dim
3746         }
3747         \bool_gset_false:N \g_@@_empty_cell_bool
3748         \skip_horizontal:n { -2 \col@sep }
3749     }
3750 }

```

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

```

3751 \tl_const:Nn \c_@@_preamble_last_col_tl
3752 {
3753     >
3754     {
3755         \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).

```

3756         \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.

```

3757         \bool_gset_true:N \g_@@_last_col_found_bool
3758         \int_gincr:N \c@jCol
3759         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3760         \hbox_set:Nw \l_@@_cell_box
3761         \@@_math_toggle:
3762         \@@_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”.

```

3763         \int_compare:nNnT \c@iRow > \c_zero_int
3764         {
3765             \bool_lazy_or:nnT
3766             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3767             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3768             {
3769                 \l_@@_code_for_last_col_tl
3770                 \xglobal \colorlet { nicematrix-last-col } { . }
3771             }
3772         }
3773     }
3774     l
3775     <

```

```

3776 {
3777   \@@_math_toggle:
3778   \hbox_set_end:
3779   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3780   \@@_adjust_size_box:
3781   \@@_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.

```

3782   \dim_gset:Nn \g_@@_width_last_col_dim
3783   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3784   \skip_horizontal:n { -2 \col@sep }

```

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

```

3785   \hbox_overlap_right:n
3786   {
3787     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3788     {
3789       \skip_horizontal:N \l_@@_right_delim_dim
3790       \skip_horizontal:N \l_@@_right_margin_dim
3791       \skip_horizontal:N \l_@@_extra_right_margin_dim
3792       \@@_node_cell:
3793     }
3794   }
3795   \bool_gset_false:N \g_@@_empty_cell_bool
3796 }
3797 }

```

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

```

3798 \NewDocumentEnvironment { NiceArray } { }
3799 {
3800   \bool_gset_false:N \g_@@_delims_bool
3801   \str_if_empty:NT \g_@@_name_env_str
3802   { \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).

```

3803   \NiceArrayWithDelims . .
3804 }
3805 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3806 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3807 {
3808   \NewDocumentEnvironment { #1 NiceArray } { }
3809   {
3810     \bool_gset_true:N \g_@@_delims_bool
3811     \str_if_empty:NT \g_@@_name_env_str
3812     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3813     \@@_test_if_math_mode:
3814     \NiceArrayWithDelims #2 #3
3815   }
3816   { \endNiceArrayWithDelims }
3817 }
3818 \@@_def_env:NNN p ( )
3819 \@@_def_env:NNN b [ ]
3820 \@@_def_env:NNN B \{ \}
3821 \@@_def_env:NNN v \vert \vert
3822 \@@_def_env:NNN V \Vert \Vert

```



## 13 The environment {NiceMatrix} and its variants

### 13.1 Definition of {pNiceMatrix}

```

3823 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3824 {
3825   \bool_set_false:N \l_@@_preamble_bool
3826   \tl_clear:N \l_tmpa_tl
3827   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3828     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3829   \tl_put_right:Nn \l_tmpa_tl
3830   {
3831     *
3832     {
3833       \int_case:nnF \l_@@_last_col_int
3834       {
3835         { -2 } { \c@MaxMatrixCols }
3836         { -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).

```

3837     }
3838     { \int_eval:n { \l_@@_last_col_int - 1 } }
3839   }
3840   { #2 }
3841 }
3842 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3843 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3844 }
3845 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3846 \clist_map_inline:nn { p , b , B , v , V }
3847 {
3848   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3849   {
3850     \bool_gset_true:N \g_@@_delims_bool
3851     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3852     \int_if_zero:nT \l_@@_last_col_int
3853     {
3854       \bool_set_true:N \l_@@_last_col_without_value_bool
3855       \int_set:Nn \l_@@_last_col_int { -1 }
3856     }
3857     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3858     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3859   }
3860   { \use:c { end #1 NiceArray } }
3861 }

```

We define also an environment {NiceMatrix}

```

3862 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3863 {
3864   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3865   \int_if_zero:nT \l_@@_last_col_int
3866   {
3867     \bool_set_true:N \l_@@_last_col_without_value_bool
3868     \int_set:Nn \l_@@_last_col_int { -1 }
3869   }
3870   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3871   \bool_lazy_or:nnT
3872     \l_@@_except_borders_bool
3873     { \clist_if_empty_p:N \l_@@_vlines_clist }
3874     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3875   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3876 }

```

```
3877 { \endNiceArray }
```

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

```
3878 \cs_new_protected:Npn \@@_NotEmpty:
3879 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

## 13.2 The key `renew-matrix`

```
3880 \cs_set_protected:Npn \@@_renew_matrix:
3881 {
3882   \tl_map_inline:nn { pvVbB }
3883   { \RenewEnvironmentCopy { ##1matrix } { ##1NiceMatrix } }
3884 }
```

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

```
3885 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3886 {
```

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`.

```
3887   \dim_compare:nNtT\l_@@_width_dim = \c_zero_dim
3888   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3889   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3890   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3891   \tl_if_empty:NF \l_@@_short_caption_tl
3892   {
3893     \tl_if_empty:NT \l_@@_caption_tl
3894     {
3895       \@@_error_or_warning:n { short-caption~without~caption }
3896       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3897     }
3898   }
3899   \tl_if_empty:NF \l_@@_label_tl
3900   {
3901     \tl_if_empty:NT \l_@@_caption_tl
3902     { \@@_error_or_warning:n { label~without~caption } }
3903   }
3904   \NewDocumentEnvironment { TabularNote } { b }
3905   {
3906     \bool_if:NTF \l_@@_in_code_after_bool
3907     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3908     {
3909       \tl_if_empty:NF \g_@@_tabularnote_tl
3910       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3911       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3912     }
3913   }
3914   { }
3915   \@@_settings_for_tabular:
3916   \NiceArray { #2 }
3917 }
3918 { \endNiceArray }
3919 \cs_new_protected:Npn \@@_settings_for_tabular:
3920 {
3921   \bool_set_true:N \l_@@_tabular_bool
3922   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3923   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3924   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3925 }
```

```

3926 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3927 {
3928   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3929   \dim_set:Nn \l_@@_width_dim { #1 }
3930   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3931   \@@_settings_for_tabular:
3932   \NiceArray { #3 }
3933 }
3934 {
3935   \endNiceArray
3936   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3937   { \@@_error:n { NiceTabularX~without~X } }
3938 }

3939 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3940 {
3941   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3942   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3943   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3944   \@@_settings_for_tabular:
3945   \NiceArray { #3 }
3946 }
3947 { \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`).

```

3948 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3949 {
3950   \bool_lazy_all:nT
3951   {
3952     { \dim_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3953     { \l_@@_hvlines_bool }
3954     { ! \g_@@_delims_bool }
3955     { ! \l_@@_except_borders_bool }
3956   }
3957   {
3958     \bool_set_true:N \l_@@_except_borders_bool
3959     \clist_if_empty:NF \l_@@_corners_clist
3960     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3961     \tl_gput_right:Nn \g_@@_rules_tl
3962     {
3963       \@@_stroke_block:nnnnn
3964       {
3965         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3966         draw = \l_@@_rules_color_tl
3967       }
3968       { 1 } { 1 } { \int_use:N \c@iRow } { \int_use:N \c@jCol }
3969     }
3970   }
3971 }

3972 \cs_new_protected:Npn \@@_after_array:
3973 {

```

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`.

```
3974 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3975 \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.

```
3976 \bool_if:NT \g_@@_last_col_found_bool
3977 { \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`.

```
3978 \bool_if:NT \l_@@_last_col_without_value_bool
3979 { \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.

```
3980 \bool_if:NT \l_@@_last_row_without_value_bool
3981 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3982 \tl_gput_right:Ne \g_@@_aux_tl
3983 {
3984   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3985   {
3986     \int_use:N \l_@@_first_row_int ,
3987     \int_use:N \c@iRow ,
3988     \int_use:N \g_@@_row_total_int ,
3989     \int_use:N \l_@@_first_col_int ,
3990     \int_use:N \c@jCol ,
3991     \int_use:N \g_@@_col_total_int
3992   }
3993 }
3994 \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`).

```
3995 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3996 {
3997   \tl_gput_right:Ne \g_@@_aux_tl
3998   {
3999     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
4000     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
4001   }
4002 }
4003 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
4004 {
4005   \tl_gput_right:Ne \g_@@_aux_tl
4006   {
4007     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
4008     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
4009     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
4010     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
4011   }
4012 }
```

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

```
4013 \@@_create_diag_nodes:
```

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

```

4014 \pgfpicture
4015 \@@_create_aliases_last:
4016 \str_if_empty:NF \l_@@_name_str \@@_create_alias_nodes:
4017 \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.

```

4018 \bool_if:NT \l_@@_parallelize_diags_bool
4019 {
4020   \int_gzero:N \g_@@_ddots_int
4021   \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.

```

4022   \dim_gzero:N \g_@@_delta_x_one_dim
4023   \dim_gzero:N \g_@@_delta_y_one_dim
4024   \dim_gzero:N \g_@@_delta_x_two_dim
4025   \dim_gzero:N \g_@@_delta_y_two_dim
4026 }
4027 \bool_set_false:N \l_@@_initial_open_bool
4028 \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.

```

4029 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

4030 \@@_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.

```

4031 \clist_if_empty:NF \l_@@_corners_clist
4032 {
4033   \bool_if:NTF \l_@@_no_cell_nodes_bool
4034   { \@@_error:n { corners~with~no~cell~nodes } }
4035   \@@_compute_corners:
4036 }

```

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`.

```

4037 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
4038   \g_@@_pos_of_blocks_seq
4039   \g_@@_future_pos_of_blocks_seq
4040 \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-*}`).

```

4041 \@@_adjust_pos_of_blocks_seq:
4042 \@@_deal_with_rounded_corners:
4043 \legacy_if:nF { measuring@ } \@@_draw_rules:
4044 \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`.

```

4045 \IfPackageLoadedT { tikz }
4046 {
4047   \tikzset
4048   {
4049     every-picture / .style =
4050     {
4051       overlay ,
4052       remember-picture ,
4053       name-prefix = \@@_env: -
4054     }
4055   }
4056 }
4057 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
4058 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
4059 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
4060 \cs_set_eq:NN \OverBrace \@@_OverBrace
4061 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
4062 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
4063 \cs_set_eq:NN \line \@@_line

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```

4064 \legacy_if:nF { measuring@ } \g_@@_pre_code_after_tl
4065 \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.

```

4066 \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).

```

4067 \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`).

```

4068 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4069 { \@@_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:`.

```

4070 \bool_set_true:N \l_@@_in_code_after_bool
4071 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4072 \scan_stop:
4073 \tl_gclear:N \g_nicematrix_code_after_tl
4074 \clist_if_empty:NF \g_@@_col_with_trees_clist \@@_draw_trees:
4075 \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.

```

4076 \seq_if_empty:NF \g_@@_rowlistcolors_seq \@@_clear_rowlistcolors_seq:
4077 \tl_if_empty:NF \g_@@_pre_code_before_tl
4078 {
4079   \tl_gput_right:Ne \g_@@_aux_tl
4080   {
4081     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4082     { \exp_not:o \g_@@_pre_code_before_tl }
4083   }
4084   \tl_gclear:N \g_@@_pre_code_before_tl

```

```

4085     }
4086     \tl_if_empty:NF \g_nicematrix_code_before_tl
4087     {
4088         \tl_gput_right:Ne \g_@@_aux_tl
4089         {
4090             \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4091             { \exp_not:o \g_nicematrix_code_before_tl }
4092         }
4093         \tl_gclear:N \g_nicematrix_code_before_tl
4094     }

4095     \str_gclear:N \g_@@_name_env_str
4096     \@@_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`.

```

4097     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4098 }

4099 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4100 {
4101     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4102     \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.

4103     \dim_set:Nn \l_@@_xdots_shorten_start_dim
4104     { 0.6 \l_@@_xdots_shorten_start_dim }
4105     \dim_set:Nn \l_@@_xdots_shorten_end_dim
4106     { 0.6 \l_@@_xdots_shorten_end_dim }
4107 }

```

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`.

```

4108 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4109 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4110 \cs_new_protected:Npn \@@_create_alias_nodes:
4111 {
4112     \int_step_inline:nn \c@iRow
4113     {
4114         \pgfnodealias
4115         { \l_@@_name_str - ##1 - last }
4116         { \@@_env: - ##1 - \int_use:N \c@jCol }
4117     }
4118     \int_step_inline:nn \c@jCol
4119     {
4120         \pgfnodealias
4121         { \l_@@_name_str - last - ##1 }
4122         { \@@_env: - \int_use:N \c@iRow - ##1 }
4123     }
4124     \pgfnodealias
4125     { \l_@@_name_str - last - last }
4126     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4127 }

```

---

<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).

```

4128 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4129 {
4130   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4131   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4132 }

```

The following command must *not* be protected.

```

4133 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4134 {
4135   { #1 }
4136   { #2 }
4137   {
4138     \int_compare:nNnTF { #3 } > { 98 }
4139     { \int_use:N \c@iRow }
4140     { #3 }
4141   }
4142   {
4143     \int_compare:nNnTF { #4 } > { 98 }
4144     { \int_use:N \c@jCol }
4145     { #4 }
4146   }
4147   { #5 }
4148 }

```

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).

```

4149 \AtBeginDocument
4150 {
4151   \cs_new_protected:Npe \@@_draw_dotted_lines:
4152   {
4153     \c_@@_pgfortikzpicture_tl
4154     \@@_draw_dotted_lines_i:
4155     \c_@@_endpgfortikzpicture_tl
4156   }
4157 }

```

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

```

4158 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4159 {
4160   \pgfrememberpicturepositiononpagetrue
4161   \pgf@relevantforpicturesizefalse
4162   \g_@@_HVdotsfor_lines_tl
4163   \g_@@_Vdots_lines_tl
4164   \g_@@_Ddots_lines_tl
4165   \g_@@_Iddots_lines_tl
4166   \g_@@_Cdots_lines_tl
4167   \g_@@_Ldots_lines_tl
4168 }

4169 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4170 {
4171   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4172   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4173 }

```



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

```

4174 \pgfdeclareshape { @@_diag_node }
4175 {
4176   \savedanchor { \five }
4177   {
4178     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4179     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4180   }
4181   \anchor { 5 } { \five }
4182   \anchor { center } { \pgfpointorigin }
4183   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4184   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4185   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4186   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4187   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4188   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4189   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4190   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4191   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4192   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4193 }

```

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).

```

4194 \cs_new_protected:Npn \@@_create_diag_nodes:
4195 {
4196   \pgfpicture
4197   \pgfrememberpicturepositiononpagetrue
4198   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4199   {
4200     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4201     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4202     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4203     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4204     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4205     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4206     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4207     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4208     \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.

```

4209     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4210     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4211     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4212     \str_if_empty:NF \l_@@_name_str
4213     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4214   }

```

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

```

4215     \int_set:Nn \l_tmpa_int { 1 + \int_max:nn \c@iRow \c@jCol } % modified
4216     \@@_qpoint:n { row - \int_min:nn \l_tmpa_int { \c@iRow + 1 } }
4217     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4218     \@@_qpoint:n { col - \int_min:nn \l_tmpa_int { \c@jCol + 1 } }
4219     \pgfcoordinate
4220     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4221     \pgfnodealias
4222     { \@@_env: - last }
4223     { \@@_env: - \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
4224     \str_if_empty:NF \l_@@_name_str
4225     {

```

```

4226     \pgfnodealias
4227     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4228     { \@@_env: - \int_use:N \l_tmpa_int }
4229     \pgfnodealias
4230     { \l_@@_name_str - last }
4231     { \@@_env: - last }
4232   }
4233 \endpgfpicture
4234 }

```

## 16 We draw the dotted lines

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

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

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

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the  $x$ -value of the orientation vector of the line;
- the fourth argument is the  $y$ -value of the orientation vector of the line.

This command computes:

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

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

```

\cs_new_protected:Npn \@@_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.

```

4235 \cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
4236 {

```

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

```

4237 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4238 \l_@@_initial_i_int = #1
4239 \l_@@_initial_j_int = #2
4240 \l_@@_final_i_int = #1
4241 \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.

```

4242 \let \l_@@_stop_loop_bool \c_false_bool
4243 \bool_do_until:Nn \l_@@_stop_loop_bool
4244 {

```

We test if we are still in the matrix.

```

4245 \advance \l_@@_final_i_int by #3
4246 \advance \l_@@_final_j_int by #4
4247 \let \l_@@_final_open_bool \c_false_bool
4248 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4249 \if_int_compare:w #3 = \c_one_int
4250 \let \l_@@_final_open_bool \c_true_bool
4251 \else:
4252 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4253 \let \l_@@_final_open_bool \c_true_bool
4254 \fi:
4255 \fi:
4256 \else:
4257 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4258 \if_int_compare:w #4 = -1
4259 \let \l_@@_final_open_bool \c_true_bool
4260 \fi:
4261 \else:
4262 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4263 \if_int_compare:w #4 = \c_one_int
4264 \let \l_@@_final_open_bool \c_true_bool
4265 \fi:
4266 \fi:
4267 \fi:
4268 \fi:
4269 \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.

```

4270 {

```

We do a step backwards.

```

4271 \advance \l_@@_final_i_int by - #3
4272 \advance \l_@@_final_j_int by - #4
4273 \let \l_@@_stop_loop_bool \c_true_bool
4274 }

```

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`.

```

4275     {
4276         \cs_if_exist:cTF
4277         {
4278             @@ _ dotted _
4279             \int_use:N \l_@@_final_i_int -
4280             \int_use:N \l_@@_final_j_int
4281         }
4282         {
4283             \advance \l_@@_final_i_int by - #3
4284             \advance \l_@@_final_j_int by - #4
4285             \let \l_@@_final_open_bool \c_true_bool
4286             \let \l_@@_stop_loop_bool \c_true_bool
4287         }
4288     {
4289         \cs_if_exist:cTF
4290         {
4291             pgf @ sh @ ns @ \@@_env:
4292             - \int_use:N \l_@@_final_i_int
4293             - \int_use:N \l_@@_final_j_int
4294         }
4295         { \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.

```

4296     {
4297         \cs_set_nopar:cpn
4298         {
4299             @@ _ dotted _
4300             \int_use:N \l_@@_final_i_int -
4301             \int_use:N \l_@@_final_j_int
4302         }
4303         { }
4304     }
4305 }
4306 }
4307 }

```

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

```

4308     \let \l_@@_stop_loop_bool \c_false_bool

```

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

```

4309     \l_tmpa_int = \l_@@_col_min_int
4310     \advance \l_tmpa_int by -1
4311     \bool_do_until:Nn \l_@@_stop_loop_bool
4312     {
4313         \advance \l_@@_initial_i_int by - #3
4314         \advance \l_@@_initial_j_int by - #4
4315         \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.

```

4316     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4317     \if_int_compare:w #3 = \c_one_int
4318         \let \l_@@_initial_open_bool \c_true_bool
4319     \else:

```

\l\_tmpa\_int contains \l\_@@\_col\_min\_int - 1 (only for efficiency).

```

4320         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4321         \let \l_@@_initial_open_bool \c_true_bool
4322     \fi:
4323 \fi:
4324 \else:
4325     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4326     \if_int_compare:w #4 = \c_one_int
4327     \let \l_@@_initial_open_bool \c_true_bool
4328     \fi:
4329 \else:
4330     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4331     \if_int_compare:w #4 = -1
4332     \let \l_@@_initial_open_bool \c_true_bool
4333     \fi:
4334 \fi:
4335 \fi:
4336 \fi:
4337 \bool_if:NTF \l_@@_initial_open_bool
4338 {
4339     \advance \l_@@_initial_i_int by #3
4340     \advance \l_@@_initial_j_int by #4
4341     \let \l_@@_stop_loop_bool \c_true_bool
4342 }
4343 {
4344     \cs_if_exist:cTF
4345     {
4346         @@ _ dotted _
4347         \int_use:N \l_@@_initial_i_int -
4348         \int_use:N \l_@@_initial_j_int
4349     }
4350     {
4351         \advance \l_@@_initial_i_int by #3
4352         \advance \l_@@_initial_j_int by #4
4353         \let \l_@@_initial_open_bool \c_true_bool
4354         \let \l_@@_stop_loop_bool \c_true_bool
4355     }
4356     {
4357         \cs_if_exist:cTF
4358         {
4359             pgf @ sh @ ns @ \@@_env:
4360             - \int_use:N \l_@@_initial_i_int
4361             - \int_use:N \l_@@_initial_j_int
4362         }
4363         { \let \l_@@_stop_loop_bool \c_true_bool }
4364         {
4365             \cs_set_nopar:cpn
4366             {
4367                 @@ _ dotted _
4368                 \int_use:N \l_@@_initial_i_int -
4369                 \int_use:N \l_@@_initial_j_int
4370             }
4371             { }
4372         }
4373     }
4374 }
4375 }
```

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.

```

4376     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4377     {
4378         { \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`.

```

4379         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4380         { \int_use:N \l_@@_final_i_int }
4381         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4382         { }
4383     }
4384 }

```

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.

```

4385 \cs_new_protected:Npn \@@_open_shorten:
4386 {
4387     \bool_if:NT \l_@@_initial_open_bool
4388     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4389     \bool_if:NT \l_@@_final_open_bool
4390     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4391 }

```

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).

```

4392 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4393 {
4394     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4395     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4396     \int_set_eq:NN \l_@@_row_max_int \c_iRow
4397     \int_set_eq:NN \l_@@_col_max_int \c_jCol

```

If there is no submatrices, we will speed up a little for the potential other dotted lines to draw.

```

4398     \seq_if_empty:NTF \g_@@_submatrix_seq
4399     { \cs_set_eq:NN \@@_adjust_to_submatrix:nn \use_none:nn }
4400     {

```

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`.

```

4401         \seq_map_inline:Nn \g_@@_submatrix_seq
4402         { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4403     }
4404 }

```

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

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

```

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

```



```

        \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
    }
}

```

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

```

4405 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4406 {
4407     \if_int_compare:w #3 > #1
4408     \else:
4409         \if_int_compare:w #1 > #5
4410         \else:
4411             \if_int_compare:w #4 > #2
4412             \else:
4413                 \if_int_compare:w #2 > #6
4414                 \else:
4415                     \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4416                     \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4417                     \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4418                     \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4419                 \fi:
4420             \fi:
4421         \fi:
4422     \fi:
4423 }

4424 \cs_new_protected:Npn \@@_set_initial_coords:
4425 {
4426     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4427     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4428 }
4429 \cs_new_protected:Npn \@@_set_final_coords:
4430 {
4431     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4432     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4433 }
4434 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4435 {
4436     \pgfpointanchor
4437     {
4438         \@@_env:
4439         - \int_use:N \l_@@_initial_i_int
4440         - \int_use:N \l_@@_initial_j_int
4441     }
4442     { #1 }
4443     \@@_set_initial_coords:
4444 }
4445 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4446 {
4447     \pgfpointanchor
4448     {
4449         \@@_env:
4450         - \int_use:N \l_@@_final_i_int
4451         - \int_use:N \l_@@_final_j_int
4452     }
4453     { #1 }
4454     \@@_set_final_coords:
4455 }
4456 \cs_new_protected:Npn \@@_open_x_initial_dim:
4457 {
4458     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4459     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4460     {
4461         \cs_if_exist:cT

```

```

4462     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4463     {
4464         \pgfpointanchor
4465         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4466         { west }
4467         \dim_set:Nn \l_@@_x_initial_dim
4468         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4469     }
4470 }

```

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

```

4471 \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4472 {
4473     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4474     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4475     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4476 }
4477 }
4478 \cs_new_protected:Npn \@@_open_x_final_dim:
4479 {
4480     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4481     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4482     {
4483         \cs_if_exist:cT
4484         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4485         {
4486             \pgfpointanchor
4487             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4488             { east }
4489             \dim_compare:nNtT \pgf@x > \l_@@_x_final_dim
4490             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4491         }
4492     }

```

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

```

4493 \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4494 {
4495     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4496     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4497     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4498 }
4499 }

```

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.

```

4500 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4501 {
4502     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4503     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4504     {
4505         \@@_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.

```

4506     \bool_if:NT \g_@@_aux_found_bool
4507     {
4508         {
4509             \@@_open_shorten:
4510             \int_if_zero:nTF { #1 }
4511             { \color { nicematrix-first-row } }
4512         }

```

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.

```

4513         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4514         { \color { nicematrix-last-row } }
4515     }
4516     \keys_set:nn { nicematrix / xdots } { #3 }
4517     \@@_color:o \l_@@_xdots_color_tl
4518     \@@_actually_draw_Ldots:
4519 }
4520 }
4521 }
4522 }
```

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`.

```

4523 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4524 {
4525     \bool_if:NTF \l_@@_initial_open_bool
4526     {
4527         \@@_open_x_initial_dim:
4528         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4529         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4530     }
4531     { \@@_set_initial_coords_from_anchor:n { base~east } }
4532     \bool_if:NTF \l_@@_final_open_bool
4533     {
4534         \@@_open_x_final_dim:
4535         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4536         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4537     }
4538     { \@@_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.

```

4539     \bool_lazy_all:nTF
4540     {
4541         \l_@@_initial_open_bool
4542         \l_@@_final_open_bool
4543         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4544     }
4545     {
4546         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4547         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4548     }
```

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

```

4549     {
4550         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4551         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4552     }
4553     \@@_draw_line:
4554 }
```

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.

```

4555 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4556 {
4557   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4558   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4559   {
4560     \@@_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.

```

4561     \bool_if:NT \g_@@_aux_found_bool
4562     {
4563       {
4564         \@@_open_shorten:
4565         \int_if_zero:nTF { #1 }
4566         { \color { nicematrix-first-row } }
4567         {

```

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.

```

4568         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4569         { \color { nicematrix-last-row } }
4570       }
4571       \keys_set:nn { nicematrix / xdots } { #3 }
4572       \@@_color:o \l_@@_xdots_color_tl
4573       \@@_actually_draw_Cdots:
4574     }
4575   }
4576 }
4577 }

```

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`.

```

4578 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4579 {
4580   \bool_if:NTF \l_@@_initial_open_bool
4581   \@@_open_x_initial_dim:
4582   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4583   \bool_if:NTF \l_@@_final_open_bool
4584   \@@_open_x_final_dim:
4585   { \@@_set_final_coords_from_anchor:n { mid-west } }
4586   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4587   {
4588     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4589     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4590     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4591     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4592     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4593   }
4594   {
4595     \bool_if:NT \l_@@_initial_open_bool
4596     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4597     \bool_if:NT \l_@@_final_open_bool

```

```

4598         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4599     }
4600     \@@_draw_line:
4601 }
4602 \cs_new_protected:Npn \@@_open_y_initial_dim:
4603 {
4604     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4605     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4606     {
4607         \cs_if_exist:cT
4608         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4609         {
4610             \pgfpointanchor
4611             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4612             { north }
4613             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4614             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4615         }
4616     }
4617     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4618     {
4619         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4620         \dim_set:Nn \l_@@_y_initial_dim
4621         {
4622             \fp_to_dim:n
4623             {
4624                 \pgf@y
4625                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4626             }
4627         }
4628     }
4629 }
4630 \cs_new_protected:Npn \@@_open_y_final_dim:
4631 {
4632     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4633     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4634     {
4635         \cs_if_exist:cT
4636         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4637         {
4638             \pgfpointanchor
4639             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4640             { south }
4641             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4642             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4643         }
4644     }
4645     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4646     {
4647         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4648         \dim_set:Nn \l_@@_y_final_dim
4649         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4650     }
4651 }

```

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.

```

4652 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4653 {
4654     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4655     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4656     {
4657         \@@_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.

```

4658     \bool_if:NT \g_@@_aux_found_bool
4659     {
4660     {
4661     \@@_open_shorten:
4662     \int_if_zero:nTF { #2 }
4663     { \color { nicematrix-first-col } }
4664     {
4665     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4666     { \color { nicematrix-last-col } }
4667     }
4668     \keys_set:nn { nicematrix / xdots } { #3 }
4669     \@@_color:o \l_@@_xdots_color_tl
4670     \bool_if:NTF \l_@@_Vbrace_bool
4671     \@@_actually_draw_Vbrace:
4672     \@@_actually_draw_Vdots:
4673     }
4674     }
4675   }
4676 }

```

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`.

```

4677 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4678 {
4679   \bool_lazy_and:nnTF \l_@@_initial_open_bool \l_@@_final_open_bool
4680   \@@_actually_draw_Vdots_i:
4681   \@@_actually_draw_Vdots_ii:
4682   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4683   \@@_draw_line:
4684 }

```

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

```

4685 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4686 {
4687   \@@_open_y_initial_dim:
4688   \@@_open_y_final_dim:
4689   \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4690   {
4691     \@@_qpoint:n { col - 1 }
4692     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4693     \dim_sub:Nn \l_@@_x_initial_dim
4694     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4695   }
4696   {
4697     \bool_lazy_and:nnTF
4698     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4699     { \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”.

```

4700     {
4701         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4702         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4703         \dim_add:Nn \l_@@_x_initial_dim
4704             { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4705     }

```

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

```

4706     {
4707         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4708         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4709         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4710         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4711     }
4712 }
4713 }

```

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.

```

4714 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4715 {
4716     \bool_set_false:N \l_tmpa_bool
4717     \bool_if:NF \l_@@_initial_open_bool
4718     {
4719         \bool_if:NF \l_@@_final_open_bool
4720         {
4721             \@@_set_initial_coords_from_anchor:n { south-west }
4722             \@@_set_final_coords_from_anchor:n { north-west }
4723             \bool_set:Nn \l_tmpa_bool
4724                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4725         }
4726     }

```

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

```

4727     \bool_if:NTF \l_@@_initial_open_bool
4728     {
4729         \@@_open_y_initial_dim:
4730         \@@_set_final_coords_from_anchor:n { north }
4731         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4732     }
4733     {
4734         \@@_set_initial_coords_from_anchor:n { south }
4735         \bool_if:NTF \l_@@_final_open_bool
4736             \@@_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.

```

4737     {
4738         \@@_set_final_coords_from_anchor:n { north }
4739         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4740         {
4741             \dim_set:Nn \l_@@_x_initial_dim
4742             {
4743                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4744                     \l_@@_x_initial_dim \l_@@_x_final_dim
4745             }
4746         }
4747     }
4748 }
4749 }

```

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`.

```

4750 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4751 {
4752   \bool_if:NTF \l_@@_initial_open_bool
4753     \@@_open_y_initial_dim:
4754     { \@@_set_initial_coords_from_anchor:n { south } }
4755   \bool_if:NTF \l_@@_final_open_bool
4756     \@@_open_y_final_dim:
4757     { \@@_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.

```

4758   \int_if_zero:nTF \l_@@_initial_j_int
4759   {
4760     \@@_qpoint:n { col - 1 }
4761     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4762     \dim_sub:Nn \l_@@_x_initial_dim
4763     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4764   }

```

Elsewhere, the brace must be drawn left flush.

```

4765   {
4766     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4767     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4768     \dim_add:Nn \l_@@_x_initial_dim
4769     { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4770   }

```

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

```

4771   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4772   \@@_draw_line:
4773 }

```

```

4774 \cs_new:Npn \@@_colsep:
4775 { \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.

```

4776 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4777 {
4778   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4779   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4780   {
4781     \@@_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.

```

4782     \bool_if:NT \g_@@_aux_found_bool
4783     {
4784     {
4785         \@@_open_shorten:
4786         \keys_set:nn { nicematrix / xdots } { #3 }
4787         \@@_color:o \l_@@_xdots_color_tl
4788         \@@_actually_draw_Ddots:
4789     }
4790     }
4791 }
4792 }

```

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`.

```

4793 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4794 {
4795     \bool_if:NTF \l_@@_initial_open_bool
4796     {
4797         \@@_open_y_initial_dim:
4798         \@@_open_x_initial_dim:
4799     }
4800     { \@@_set_initial_coords_from_anchor:n { south~east } }
4801     \bool_if:NTF \l_@@_final_open_bool
4802     {
4803         \@@_open_x_final_dim:
4804         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4805     }
4806     { \@@_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.

```

4807     \bool_if:NT \l_@@_parallelize_diags_bool
4808     {
4809         \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).

```

4810         \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.

```

4811     {
4812         \dim_gset:Nn \g_@@_delta_x_one_dim
4813         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4814         \dim_gset:Nn \g_@@_delta_y_one_dim
4815         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4816     }

```

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`.

```

4817     {
4818         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4819         {
4820             \dim_set:Nn \l_@@_y_final_dim
4821             {
4822                 \l_@@_y_initial_dim +
4823                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4824                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4825             }
4826         }
4827     }
4828 }
4829 \@@_draw_line:
4830 }

```

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.

```

4831 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4832 {
4833     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4834     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4835     {
4836         \@@_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.

```

4837         \bool_if:NT \g_@@_aux_found_bool
4838         {
4839             {
4840                 \@@_open_shorten:
4841                 \keys_set:nn { nicematrix / xdots } { #3 }
4842                 \@@_color:o \l_@@_xdots_color_tl
4843                 \@@_actually_draw_Iddots:
4844             }
4845         }
4846     }
4847 }

```

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`.

```

4848 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4849 {
4850     \bool_if:NTF \l_@@_initial_open_bool
4851     {
4852         \@@_open_y_initial_dim:
4853         \@@_open_x_initial_dim:
4854     }
4855     { \@@_set_initial_coords_from_anchor:n { south-west } }
4856     \bool_if:NTF \l_@@_final_open_bool

```

```

4857 {
4858   \l_@@_open_y_final_dim:
4859   \l_@@_open_x_final_dim:
4860 }
4861 { \l_@@_set_final_coords_from_anchor:n { north-east } }
4862 \bool_if:NT \l_@@_parallelize_diags_bool
4863 {
4864   \int_gincr:N \g_@@_iddots_int
4865   \int_compare:nNnTF \g_@@_iddots_int = 1
4866   {
4867     \dim_gset:Nn \g_@@_delta_x_two_dim
4868     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4869     \dim_gset:Nn \g_@@_delta_y_two_dim
4870     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4871   }
4872   {
4873     \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4874     {
4875       \dim_set:Nn \l_@@_y_final_dim
4876       {
4877         \l_@@_y_initial_dim +
4878         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4879         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4880       }
4881     }
4882   }
4883 }
4884 \l_@@_draw_line:
4885 }

```

## 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`

```

4886 \cs_new_protected:Npn \l_@@_draw_line:
4887 {
4888   \pgfrememberpicturepositiononpagetrue
4889   \pgf@relevantforpicturesizefalse
4890   \bool_lazy_or:nnTF
4891     \l_@@_dotted_bool
4892     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4893     \l_@@_draw_standard_dotted_line:
4894     \l_@@_draw_unstandard_dotted_line:
4895 }

```

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.

```

4896 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4897 {
4898   \begin { scope }
4899   \@@_draw_unstandard_dotted_line:o
4900   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4901 }

```

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.

```

4902 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4903 {
4904   \@@_draw_unstandard_dotted_line:nooo
4905   { #1 }
4906   \l_@@_xdots_up_tl
4907   \l_@@_xdots_down_tl
4908   \l_@@_xdots_middle_tl
4909 }
4910 \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.

```

4911 \AtBeginDocument
4912 {
4913   \IfPackageLoadedT { tikz }
4914   {
4915     \tikzset
4916     {
4917       @@_node_above / .style = { sloped , above } ,
4918       @@_node_below / .style = { sloped , below } ,
4919       @@_node_middle / .style =
4920       {
4921         sloped ,
4922         inner-sep = \c_@@_innersep_middle_dim
4923       }
4924     }
4925   }
4926 }

4927 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4928 {

```

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.

```

4929 \dim_zero_new:N \l_@@_l_dim
4930 \dim_set:Nn \l_@@_l_dim
4931 {
4932   \fp_to_dim:n
4933   {
4934     sqrt
4935     (
4936       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4937       +
4938       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4939     )
4940   }
4941 }

```

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.

```

4942   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4943   {
4944     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4945     \@@_draw_unstandard_dotted_line_i:
4946   }

```

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

```

4947   \bool_if:NT \l_@@_xdots_h_labels_bool
4948   {
4949     \tikzset
4950     {
4951       @@_node_above / .style = { auto = left } ,
4952       @@_node_below / .style = { auto = right } ,
4953       @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4954     }
4955   }
4956   \tl_if_empty:nF { #4 }
4957   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4958   \dim_zero:N \l_tmpa_dim
4959   \dim_zero:N \l_tmpb_dim
4960   \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4961   {

```

We test whether the brace is vertical or horizontal.

```

4962     \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4963     { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4964     { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4965   }
4966   {
4967     \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4968     {
4969       \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
4970       { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4971       { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4972     }
4973   }
4974   \use:e
4975   {
4976     \exp_not:N \begin { scope }
4977     [ shift = {(\dim_use:N \l_tmpa_dim, \dim_use:N \l_tmpb_dim)} ]
4978   }
4979   \draw
4980   [ #1 ]
4981   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4982   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4983   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4984   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4985   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4986   \end { scope }
4987   \end { scope }
4988 }
4989 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4990 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4991 {
4992   \dim_set:Nn \l_tmpa_dim
4993   {
4994     \l_@@_x_initial_dim
4995     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4996     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4997     }
4998     \dim_set:Nn \l_tmpb_dim
4999     {
5000         \l_@@_y_initial_dim
5001         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5002         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
5003     }
5004     \dim_set:Nn \l_@@_tmpc_dim
5005     {
5006         \l_@@_x_final_dim
5007         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
5008         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5009     }
5010     \dim_set:Nn \l_@@_tmpd_dim
5011     {
5012         \l_@@_y_final_dim
5013         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
5014         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
5015     }
5016     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
5017     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
5018     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
5019     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
5020 }

```

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

```

5021 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
5022 {
5023 {

```

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.

```

5024     \dim_zero_new:N \l_@@_l_dim
5025     \dim_set:Nn \l_@@_l_dim
5026     {
5027         \fp_to_dim:n
5028         {
5029             sqrt
5030             (
5031                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
5032                 +
5033                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
5034             )
5035         }
5036     }

```

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.

```

5037     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
5038     {
5039         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
5040         \@@_draw_standard_dotted_line_i:
5041     }
5042 }
5043 \bool_lazy_all:nF
5044 {
5045     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
5046     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
5047     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
5048 }

```

```

5049 \@@_labels_standard_dotted_line:
5050 }
5051 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
5052 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
5053 {

```

The number of dots will be  $\text{\l\_tmpa\_int} + 1$ .

```

5054 \int_set:Nn \l_tmpa_int
5055 {
5056 \dim_ratio:nn
5057 {
5058 \l_@@_l_dim
5059 - \l_@@_xdots_shorten_start_dim
5060 - \l_@@_xdots_shorten_end_dim
5061 }
5062 \l_@@_xdots_inter_dim
5063 }

```

The dimensions  $\text{\l\_tmpa\_dim}$  and  $\text{\l\_tmpb\_dim}$  are the coordinates of the vector between two dots in the dotted line.

```

5064 \dim_set:Nn \l_tmpa_dim
5065 {
5066 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5067 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5068 }
5069 \dim_set:Nn \l_tmpb_dim
5070 {
5071 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5072 \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5073 }

```

In the loop over the dots, the dimensions  $\text{\l_@@_x\_initial\_dim}$  and  $\text{\l_@@_y\_initial\_dim}$  will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

5074 \dim_gadd:Nn \l_@@_x_initial_dim
5075 {
5076 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5077 \dim_ratio:nn
5078 {
5079 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5080 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5081 }
5082 { 2 \l_@@_l_dim }
5083 }
5084 \dim_gadd:Nn \l_@@_y_initial_dim
5085 {
5086 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5087 \dim_ratio:nn
5088 {
5089 \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5090 + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5091 }
5092 { 2 \l_@@_l_dim }
5093 }
5094 \pgf@relevantforpicturesizefalse
5095 \int_step_inline:nnn \c_zero_int \l_tmpa_int
5096 {
5097 \pgfpathcircle
5098 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5099 \l_@@_xdots_radius_dim
5100 \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5101 \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5102 }
5103 \pgfusepathqfill
5104 }

```

```

5105 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5106 {
5107   \pgfscope
5108   \pgftransformshift
5109   {
5110     \pgfpointlineattime { 0.5 }
5111     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5112     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5113   }
5114   \fp_set:Nn \l_tmpa_fp
5115   {
5116     atand
5117     (
5118       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5119       \l_@@_x_final_dim - \l_@@_x_initial_dim
5120     )
5121   }
5122   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5123   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5124   \tl_if_empty:NF \l_@@_xdots_middle_tl
5125   {
5126     \begin { pgfscope }
5127     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5128     \pgfnode
5129     { rectangle }
5130     { center }
5131     {
5132       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5133       {
5134         $ % $
5135         \scriptstyle \l_@@_xdots_middle_tl
5136         $ % $
5137       }
5138     }
5139     { }
5140     {
5141       \pgfsetfillcolor { white }
5142       \pgfusepath { fill }
5143     }
5144     \end { pgfscope }
5145   }
5146   \tl_if_empty:NF \l_@@_xdots_up_tl
5147   {
5148     \pgfnode
5149     { rectangle }
5150     { south }
5151     {
5152       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5153       {
5154         $ % $
5155         \scriptstyle \l_@@_xdots_up_tl
5156         $ % $
5157       }
5158     }
5159     { }
5160     { \pgfusepath { } }
5161   }
5162   \tl_if_empty:NF \l_@@_xdots_down_tl
5163   {
5164     \pgfnode
5165     { rectangle }
5166     { north }
5167     {

```



```

5168         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5169         {
5170             $ % $
5171             \scriptstyle \l_@@_xdots_down_tl
5172             $ % $
5173         }
5174     }
5175     { }
5176     { \pgfusepath { } }
5177 }
5178 \endpgfscope
5179 }

```

## 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.

```

5180 \AtBeginDocument
5181 {

```

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

```

5182     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } { } } }
5183     \cs_new_protected:Npn \@@_Ldots: { \@@_collect_options:n { \@@_Ldots_i } }
5184     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5185     {
5186         \int_if_zero:nTF \c@jCol
5187         { \@@_error:nn { in~first~col } { \Ldots } }
5188         {
5189             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5190             { \@@_error:nn { in~last~col } { \Ldots } }
5191             {
5192                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5193                 { #1 , down = #2 , up = #3 , middle = #4 }
5194             }
5195         }
5196         \bool_if:NF \l_@@_nullify_dots_bool
5197         { \phantom { \ensuremath { \@@_old_ldots: } } }
5198         \bool_gset_true:N \g_@@_empty_cell_bool
5199     }

```

```

5200     \cs_new_protected:Npn \@@_Cdots: { \@@_collect_options:n { \@@_Cdots_i } }
5201     \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5202     {
5203         \int_if_zero:nTF \c@jCol
5204         { \@@_error:nn { in~first~col } { \Cdots } }
5205         {
5206             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5207             { \@@_error:nn { in~last~col } { \Cdots } }
5208             {
5209                 \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }

```

```

5210         { #1 , down = #2 , up = #3 , middle = #4 }
5211     }
5212 }
5213 \bool_if:NF \l_@@_nullify_dots_bool
5214 { \phantom { \ensuremath { \@@_old_cdots: } } }
5215 \bool_gset_true:N \g_@@_empty_cell_bool
5216 }

5217 \cs_new_protected:Npn \@@_Vdots: { \@@_collect_options:n { \@@_Vdots_i } }
5218 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5219 {
5220     \int_if_zero:nTF \c@iRow
5221     { \@@_error:nn { in~first~row } { \Vdots } }
5222     {
5223         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5224         { \@@_error:nn { in~last~row } { \Vdots } }
5225         {
5226             \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5227             { #1 , down = #2 , up = #3 , middle = #4 }
5228         }
5229     }
5230     \bool_if:NF \l_@@_nullify_dots_bool
5231     { \phantom { \ensuremath { \@@_old_vdots: } } }
5232     \bool_gset_true:N \g_@@_empty_cell_bool
5233 }

5234 \cs_new_protected:Npn \@@_Ddots: { \@@_collect_options:n { \@@_Ddots_i } }
5235 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5236 {
5237     \int_case:nnF \c@iRow
5238     {
5239         0 { \@@_error:nn { in~first~row } { \Ddots } }
5240         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5241     }
5242     {
5243         \int_case:nnF \c@jCol
5244         {
5245             0 { \@@_error:nn { in~first~col } { \Ddots } }
5246             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5247         }
5248         {
5249             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5250             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
5251             { #1 , down = #2 , up = #3 , middle = #4 }
5252         }
5253     }
5254 }
5255 \bool_if:NF \l_@@_nullify_dots_bool
5256 { \phantom { \ensuremath { \@@_old_ddots: } } }
5257 \bool_gset_true:N \g_@@_empty_cell_bool
5258 }

5259 \cs_new_protected:Npn \@@_Iddots:
5260 { \@@_collect_options:n { \@@_Iddots_i } }
5261 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5262 {
5263     \int_case:nnF \c@iRow
5264     {
5265         0 { \@@_error:nn { in~first~row } { \Iddots } }
5266         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5267     }

```

```

5268     {
5269         \int_case:nnF \c@jCol
5270         {
5271             0 { \@@_error:nn { in~first~col } { \Iddots } }
5272             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5273         }
5274         {
5275             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5276             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5277             { #1 , down = #2 , up = #3 , middle = #4 }
5278         }
5279     }
5280     \bool_if:NF \l_@@_nullify_dots_bool
5281     { \phantom { \ensuremath { \@@_old_iddots: } } }
5282     \bool_gset_true:N \g_@@_empty_cell_bool
5283 }
5284 }

```

End of the \AddToHook.

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

```

5285 \keys_define:nn { nicematrix / Ddots }
5286 {
5287     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5288     draw-first .value_forbidden:n = true ,
5289 }

```

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

```

5290 \cs_new_protected:Npn \@@_Hspace:
5291 {
5292     \bool_gset_true:N \g_@@_empty_cell_bool
5293     \hspace
5294 }

```

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.

```

5295 \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.

```

5296 \cs_new:Npn \@@_Hdotsfor:
5297 {
5298     \bool_lazy_and:nnTF
5299     { \int_if_zero_p:n \c@jCol }
5300     { \int_if_zero_p:n \l_@@_first_col_int }
5301     {
5302         \bool_if:NTF \g_@@_after_col_zero_bool
5303         {
5304             \multicolumn { 1 } { c } { }
5305             \@@_Hdotsfor_i:
5306         }
5307         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5308     }
5309     {
5310         \multicolumn { 1 } { c } { }
5311         \@@_Hdotsfor_i:
5312     }
5313 }

```

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:`).

```
5314 \AtBeginDocument
5315 {
```

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

```
5316 \cs_new_protected:Npn \@@_Hdotsfor_i:
5317 { \@@_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`).

```
5318 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
5319 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5320 {
5321   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5322   {
5323     \@@_Hdotsfor:nnnn
5324     { \int_use:N \c@iRow }
5325     { \int_use:N \c@jCol }
5326     { #2 }
5327     {
5328       #1 , #3 ,
5329       down = \exp_not:n { #4 } ,
5330       up = \exp_not:n { #5 } ,
5331       middle = \exp_not:n { #6 }
5332     }
5333   }
5334   \prg_replicate:nn { #2 - 1 }
5335   {
5336     &
5337     \multicolumn { 1 } { c } { }
5338     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5339   }
5340 }
5341 }
```

```
5342 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5343 {
5344   \bool_set_false:N \l_@@_initial_open_bool
5345   \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5346 \int_set:Nn \l_@@_initial_i_int { #1 }
5347 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5348 \int_compare:nNnTF { #2 } = 1
5349 {
5350   \int_set:Nn \l_@@_initial_j_int 1
5351   \bool_set_true:N \l_@@_initial_open_bool
5352 }
5353 {
5354   \cs_if_exist:cTF
5355   {
5356     pgf @ sh @ ns @ \@@_env:
5357     - \int_use:N \l_@@_initial_i_int
5358     - \int_eval:n { #2 - 1 }
5359   }
5360   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5361   {
5362     \int_set:Nn \l_@@_initial_j_int { #2 }
5363     \bool_set_true:N \l_@@_initial_open_bool
```

```

5364     }
5365   }
5366   \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5367   {
5368     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5369     \bool_set_true:N \l_@@_final_open_bool
5370   }
5371   {
5372     \cs_if_exist:cTF
5373     {
5374       pgf @ sh @ ns @ \@@_env:
5375       - \int_use:N \l_@@_final_i_int
5376       - \int_eval:n { #2 + #3 }
5377     }
5378     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5379     {
5380       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5381       \bool_set_true:N \l_@@_final_open_bool
5382     }
5383   }
5384   \bool_if:NT \g_@@_aux_found_bool
5385   {
5386     {
5387       \@@_open_shorten:
5388       \int_if_zero:nTF { #1 }
5389       { \color { nicematrix-first-row } }
5390       {
5391         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5392         { \color { nicematrix-last-row } }
5393       }
5394       \keys_set:nn { nicematrix / xdots } { #4 }
5395       \@@_color:o \l_@@_xdots_color_tl
5396       \@@_actually_draw_Ldots:
5397     }
5398   }

```

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).

```

5399   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5400   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5401 }

```

```

5402 \AtBeginDocument
5403 {
5404   \cs_new_protected:Npn \@@_Vdotsfor:
5405   { \@@_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`).

```

5406   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5407   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5408   {
5409     \bool_gset_true:N \g_@@_empty_cell_bool
5410     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5411     {
5412       \@@_Vdotsfor:nnnn
5413       { \int_use:N \c@iRow }
5414       { \int_use:N \c@jCol }
5415       { #2 }
5416       {
5417         #1 , #3 ,

```

```

5418         down = \exp_not:n { #4 } ,
5419         up = \exp_not:n { #5 } ,
5420         middle = \exp_not:n { #6 }
5421     }
5422 }
5423 }
5424 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5425 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5426 {
5427     \bool_set_false:N \l_@@_initial_open_bool
5428     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5429     \int_set:Nn \l_@@_initial_j_int { #2 }
5430     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5431     \int_compare:nNnTF { #1 } = 1
5432     {
5433         \int_set:Nn \l_@@_initial_i_int 1
5434         \bool_set_true:N \l_@@_initial_open_bool
5435     }
5436     {
5437         \cs_if_exist:cTF
5438         {
5439             pgf @ sh @ ns @ \@@_env:
5440             - \int_eval:n { #1 - 1 }
5441             - \int_use:N \l_@@_initial_j_int
5442         }
5443         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5444         {
5445             \int_set:Nn \l_@@_initial_i_int { #1 }
5446             \bool_set_true:N \l_@@_initial_open_bool
5447         }
5448     }
5449     \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5450     {
5451         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5452         \bool_set_true:N \l_@@_final_open_bool
5453     }
5454     {
5455         \cs_if_exist:cTF
5456         {
5457             pgf @ sh @ ns @ \@@_env:
5458             - \int_eval:n { #1 + #3 }
5459             - \int_use:N \l_@@_final_j_int
5460         }
5461         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5462         {
5463             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5464             \bool_set_true:N \l_@@_final_open_bool
5465         }
5466     }
5467     \bool_if:NT \g_@@_aux_found_bool
5468     {
5469         {
5470             \@@_open_shorten:
5471             \int_if_zero:nTF { #2 }
5472             { \color { nicematrix-first-col } }

```

```

5473         {
5474             \int_compare:nNtT { #2 } = \g_@@_col_total_int
5475             { \color { nicematrix-last-col } }
5476         }
5477         \keys_set:nn { nicematrix / xdots } { #4 }
5478         \@@_color:o \l_@@_xdots_color_tl
5479         \bool_if:NTF \l_@@_Vbrace_bool
5480             \@@_actually_draw_Vbrace:
5481             \@@_actually_draw_Vdots:
5482     }
5483 }

```

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).

```

5484     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5485     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5486 }

```

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

```

5487 \NewDocumentCommand \@@_rotate: { 0 { } }
5488 {
5489     \bool_gset_true:N \g_@@_rotate_bool
5490     \keys_set:nn { nicematrix / rotate } { #1 }
5491     \ignorespaces
5492 }

```

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

```

5493 \cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }

5494 \keys_define:nn { nicematrix / rotate }
5495 {
5496     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5497     c .value_forbidden:n = true ,
5498     -90 .code:n = \bool_gset_true:N \g_@@_rotate_minus_bool ,
5499     -90 .value_forbidden:n = true ,
5500     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5501 }

```

## 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>

```

5502 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5503 {
5504   \tl_if_empty:nTF { #2 }
5505     { #1 }
5506     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5507 }
5508 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5509 { \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).

```

5510 \AtBeginDocument
5511 {

```

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

```

5512   \tl_set_rescan:Nnn \l_tmpa_tl { }
5513   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5514   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5515   {
5516     {
5517       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5518       \@@_color:o \l_@@_xdots_color_tl
5519       \use:e
5520       {
5521         \@@_line_i:nn
5522         { \@@_double_int_eval:n #2 - \q_stop }
5523         { \@@_double_int_eval:n #3 - \q_stop }
5524       }
5525     }
5526   }
5527 }

5528 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5529 {
5530   \bool_set_false:N \l_@@_initial_open_bool
5531   \bool_set_false:N \l_@@_final_open_bool
5532   \bool_lazy_or:nnTF
5533     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5534     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5535     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5536   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5537 }

5538 \AtBeginDocument
5539 {
5540   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5541   {

```

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:`.

```

5542   \c_@@_pgfortikzpicture_tl
5543   \@@_draw_line_iii:nn { #1 } { #2 }
5544   \c_@@_endpgfortikzpicture_tl
5545 }
5546 }

```

---

<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`).

```

5547 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5548 {
5549   \pgfrememberpicturepositiononpagetrue
5550   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5551   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5552   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5553   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5554   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5555   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5556   \@@_draw_line:
5557 }

```

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.

```

5558 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT \c@iRow < }
5559 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF \c@jCol < }

```

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

```

5560 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5561 {
5562   \tl_gput_right:Ne \g_@@_row_style_tl
5563   {

```

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

```

5564   \exp_not:N
5565   \@@_if_row_less_than:nn
5566   { \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`).

```

5567   {
5568     \exp_not:N
5569     \@@_if_col_greater_than:nn
5570     { \int_use:N \c@jCol }
5571     { \exp_not:n { #1 } \scan_stop: }
5572   }
5573 }
5574 }
5575 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5576 \keys_define:nn { nicematrix / RowStyle }
5577 {
5578   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5579   cell-space-top-limit+ .code:n =
5580     \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
5581   cell-space-top-limit+ .value_required:n = true ,
5582   cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
5583   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5584   cell-space-bottom-limit+ .code:n =
5585     \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
5586   cell-space-bottom-limit+ .value_required:n = true ,
5587   cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
5588   cell-space-limits .meta:n =
5589     {
5590       cell-space-top-limit = #1 ,
5591       cell-space-bottom-limit = #1 ,
5592     } ,
5593   cell-space-limits+ .meta:n =
5594     {
5595       cell-space-top-limit += #1 ,
5596       cell-space-bottom-limit += #1 ,
5597     } ,
5598   cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
5599   color .tl_set:N = \l_@@_color_tl ,
5600   color .value_required:n = true ,
5601   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5602   nb-rows .code:n =
5603     \str_if_eq:eeTF { #1 } { * }
5604     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5605     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5606   nb-rows .value_required:n = true ,
5607   fill .tl_set:N = \l_@@_fill_tl ,
5608   fill .value_required:n = true ,

```

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

```

5609   opacity .tl_set:N = \l_@@_opacity_tl ,
5610   opacity .value_required:n = true ,
5611   rowcolor .tl_set:N = \l_@@_fill_tl ,
5612   rowcolor .value_required:n = true ,
5613   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5614   rounded-corners .default:n = 4 pt ,
5615   unknown .code:n =
5616     \@@_unknown_key:nn
5617     { nicematrix / RowStyle }
5618     { Unknown-key-for-RowStyle }
5619 }

```

```

5620 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5621 {
5622   \group_begin:
5623   \tl_clear:N \l_@@_fill_tl
5624   \tl_clear:N \l_@@_opacity_tl
5625   \tl_clear:N \l_@@_color_tl
5626   \int_set:Nn \l_@@_key_nb_rows_int 1
5627   \dim_zero:N \l_@@_rounded_corners_dim
5628   \dim_zero:N \l_tmpa_dim
5629   \dim_zero:N \l_tmpb_dim
5630   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5631   \tl_if_empty:NF \l_@@_fill_tl
5632   {
5633     \@@_add_opacity_to_fill:

```

```

5634 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5635 {

```

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

```

5636 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5637 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5638 {
5639 \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5640 - *
5641 }
5642 { \dim_use:N \l_@@_rounded_corners_dim }
5643 }
5644 }
5645 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5646 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5647 {
5648 \@@_put_in_row_style:e
5649 {
5650 \@@_put_in_cell_after_hook:n
5651 {

```

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

```

5652 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5653 { \dim_use:N \l_tmpa_dim }
5654 }
5655 }
5656 }

```

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

```

5657 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5658 {
5659 \@@_put_in_row_style:e
5660 {
5661 \@@_put_in_cell_after_hook:n
5662 {
5663 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5664 { \dim_use:N \l_tmpb_dim }
5665 }
5666 }
5667 }

```

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

```

5668 \tl_if_empty:NF \l_@@_color_tl
5669 {
5670 \@@_put_in_row_style:e
5671 {
5672 \mode_leave_vertical:
5673 \@@_color:n { \l_@@_color_tl }
5674 }
5675 }

```

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

```

5676 \bool_if:NT \l_@@_bold_row_style_bool
5677 {
5678 \@@_put_in_row_style:n
5679 {
5680 \exp_not:n
5681 {
5682 \if_mode_math:
5683 $ % $
5684 \bfseries \boldmath
5685 $ % $
5686 \else:
5687 \bfseries \boldmath

```

```

5688         \fi:
5689     }
5690 }
5691 }
5692 \group_end:
5693 \g_@@_row_style_tl
5694 \ignorespaces
5695 }

```

The following commande must *not* be protected.

```

5696 \cs_new:Npn \@@_rounded_from_row:n #1
5697 {
5698     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5699     { \int_eval:n { #1 } - 1 }
5700     {
5701         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5702         - \exp_not:n { \int_use:N \c@jCol }
5703     }
5704     { \dim_use:N \l_@@_rounded_corners_dim }
5705 }

```

## 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).

```

5706 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5707 {

```

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.

```

5708     \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`.

```

5709     \str_if_in:nnF { #1 } { !! }
5710     {
5711         \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

5712     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5713   }
5714   \int_if_zero:nTF \l_tmpa_int

```

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

```

5715   {
5716     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5717     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5718   }

```

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

```

5719     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5720   }
5721   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }

```

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

```

5722   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5723   {
5724     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5725     {

```

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

```

5726     \group_begin:
5727     \pgfsetcornersarced
5728     { \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.

```

5729     \bool_if:NTF \l_@@_hvlines_bool
5730     {
5731       \pgfpathrectanglecorners
5732       {
5733         \pgfpointadd
5734         { \@@_qpoint:n { row-1 } }
5735         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
5736       }
5737       {
5738         \pgfpointadd
5739         {
5740           \@@_qpoint:n
5741           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5742         }
5743         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5744       }
5745     }
5746     {
5747       \pgfpathrectanglecorners
5748       { \@@_qpoint:n { row-1 } }
5749       {
5750         \pgfpointadd
5751         {
5752           \@@_qpoint:n
5753           { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } }
5754         }
5755         { \pgfpoint \c_zero_dim \arrayrulewidth }
5756       }
5757     }
5758     \pgfusepath { clip }
5759     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```
5760     }
5761 }
```

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

```
5762 \cs_new_protected:Npn \@@_actually_color:
5763 {
5764   \pgfpicture
5765   \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.

```
5766   \@@_clip_with_rounded_corners:
```

We will now actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```
5767   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5768   {
5769     \int_compare:nNnTF { ##1 } = 1
5770     {
5771       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5772       \use:c { g_@@_color _ 1 _tl }
5773       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5774     }
5775     {
5776       \pgfscope
5777       \@@_color_opacity: ##2
5778       \use:c { g_@@_color _ ##1 _tl }
5779       \tl_gclear:c { g_@@_color _ ##1 _tl }
5780       \pgfusepath { fill }
5781       \endpgfscope
5782     }
5783   }
5784   \endpgfpicture
5785 }
```

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

```
5786 \cs_new_protected:Npn \@@_color_opacity:
5787 {
5788   \peek_meaning:NTF [
5789     \@@_color_opacity:w
5790     { \@@_color_opacity:w [ ] }
5791   }
```

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.

```
5792 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5793 {
5794   \tl_clear:N \l_tmpa_tl
5795   \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.

```
5796   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5797   \tl_if_empty:NTF \l_tmpb_tl
5798   \@declaredcolor
5799   { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5800 }
```

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

```

5801 \keys_define:nn { nicematrix / color-opacity }
5802 {
5803   opacity .tl_set:N          = \l_tmpa_tl ,
5804   opacity .value_required:n = true
5805 }

```

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

```

5806 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5807 {
5808   \def \l_@@_rows_tl { #1 }
5809   \def \l_@@_cols_tl { #2 }
5810   \@@_cartesian_path:
5811 }

```

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

```

5812 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5813 {
5814   \tl_if_blank:nF { #2 }
5815   {
5816     \@@_add_to_colors_seq:en
5817     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5818     { \@@_cartesian_color:nn { #3 } { - } }
5819   }
5820 }

```

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

```

5821 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5822 {
5823   \tl_if_blank:nF { #2 }
5824   {
5825     \@@_add_to_colors_seq:en
5826     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5827     { \@@_cartesian_color:nn { - } { #3 } }
5828   }
5829 }

```

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

```

5830 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5831 {
5832   \tl_if_blank:nF { #2 }
5833   {
5834     \@@_add_to_colors_seq:en
5835     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5836     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5837   }
5838 }

```

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

```

5839 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5840 {
5841   \tl_if_blank:nF { #2 }
5842   {
5843     \@@_add_to_colors_seq:en
5844     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5845     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5846   }
5847 }

```

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

```

5848 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5849 {
5850   \@@_cut_on_hyphen:w #1 \q_stop
5851   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5852   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5853   \@@_cut_on_hyphen:w #2 \q_stop
5854   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5855   \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`.

```

5856   \@@_cartesian_path:n { #3 }
5857 }

```

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

```

5858 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5859 {
5860   \clist_map_inline:nn { #3 }
5861     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5862 }

```

```

5863 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5864 {
5865   \int_step_inline:nn \c@iRow
5866   {
5867     \int_step_inline:nn \c@jCol
5868     {
5869       \int_if_even:nTF { ####1 + ##1 }
5870         { \@@_cellcolor [ #1 ] { #2 } }
5871         { \@@_cellcolor [ #1 ] { #3 } }
5872       { ##1 - ####1 }
5873     }
5874   }
5875 }

```

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”.

```

5876 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5877 {
5878   \@@_rectanglecolor [ #1 ] { #2 }
5879   { 1 - 1 }
5880   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5881 }

5882 \keys_define:nn { nicematrix / rowcolors }
5883 {
5884   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5885   cols .tl_set:N = \l_@@_cols_tl ,
5886   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5887   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5888 }

```

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*.

```
5889 \NewDocumentCommand \l_@@_rowlistcolors { 0 { } m m 0 { } }
5890 {
```

\l\_@@\_colors\_seq will be the list of colors.

```
5891 \seq_clear_new:N \l_@@_colors_seq
5892 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5893 \tl_clear_new:N \l_@@_cols_tl
5894 \tl_set:Nn \l_@@_cols_tl { - }
5895 \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).

```
5896 \int_zero_new:N \l_@@_color_int
5897 \int_set:Nn \l_@@_color_int 1
5898 \bool_if:NT \l_@@_respect_blocks_bool
5899 {
```

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).

```
5900 % modified 2026-02-18
5901 \seq_set_filter:NNn \l_tmpa_seq \g_@@_pos_of_blocks_seq
5902 { \l_@@_not_in_exterior_p:nnnnn ##1 }
5903 }
```

#2 is the list of intervals of rows.

```
5904 \clist_map_inline:nn { #2 }
5905 {
5906 \tl_set:Nn \l_tmpa_tl { ##1 }
5907 \tl_if_in:NnTF \l_tmpa_tl { - }
5908 { \l_@@_cut_on_hyphen:w ##1 \q_stop }
5909 { \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.

```
5910 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5911 \int_set:Nn \l_@@_color_int
5912 { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } \l_tmpa_tl }
5913 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5914 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5915 {
```

We will compute in \l\_tmpb\_int the last row of the “block”.

```
5916 \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).

```
5917 \bool_if:NT \l_@@_respect_blocks_bool
5918 {
5919 \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5920 { \l_@@_intersect_our_row_p:nnnnn #####1 }
5921 \seq_map_inline:Nn \l_tmpb_seq { \l_@@_rowcolors_i:nnnnn #####1 }
```

Now, the last row of the block is computed in \l\_tmpb\_int.

```
5922 }
5923 \tl_set:Nn \l_@@_rows_tl
5924 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

\l\_@@\_tmpc\_tl will be the color that we will use.

```
5925 \tl_set:Nn \l_@@_color_tl
5926 {
5927 \l_@@_color_index:n
5928 {
5929 \int_mod:nn
5930 { \l_@@_color_int - 1 }
5931 { \seq_count:N \l_@@_colors_seq }
```

```

5932         + 1
5933     }
5934 }
5935 \tl_if_empty:NF \l_@@_color_tl
5936 {
5937     \use:e
5938     {
5939         \@@_add_to_colors_seq:nn
5940         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5941         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5942     }
5943 }
5944 \int_incr:N \l_@@_color_int
5945 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5946 }
5947 }
5948 }

```

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.

```

5949 \cs_new:Npn \@@_color_index:n #1
5950 {

```

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

```

5951     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5952     { \@@_color_index:n { #1 - 1 } }
5953     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5954 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5955 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5956 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5957 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5958 {
5959     \int_compare:nNnT { #3 } > \l_tmpb_int
5960     { \int_set:Nn \l_tmpb_int { #3 } }
5961 }

```

```

5962 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5963 {
5964     \int_if_zero:nTF { #4 }
5965     \prg_return_false:
5966     {
5967         \int_compare:nNnTF { #2 } > \c@jCol
5968         \prg_return_false:
5969         \prg_return_true:
5970     }
5971 }

```

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

```

5972 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5973 {
5974     \int_compare:nNnTF { #1 } > \l_tmpa_int
5975     \prg_return_false:
5976     {
5977         \int_compare:nNnTF \l_tmpa_int > { #3 }

```

```

5978         \prg_return_false:
5979         \prg_return_true:
5980     }
5981 }

```

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`).

```

5982 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5983 {
5984     \dim_compare:nNnTF { #1 } = \c_zero_dim
5985     {
5986         \bool_if:NTF \l_@@_nocolor_used_bool
5987         { \@@_cartesian_path_normal_ii: }
5988         {
5989             \clist_if_empty:NTF \l_@@_corners_cells_clist
5990             { \@@_cartesian_path_normal_i:n { #1 } }
5991             { \@@_cartesian_path_normal_ii: }
5992         }
5993     }
5994     { \@@_cartesian_path_normal_i:n { #1 } }
5995 }

```

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.

```

5996 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5997 {
5998     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5999     \clist_map_inline:Nn \l_@@_cols_tl
6000     {

```

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

```

6001         \def \l_tmpa_tl { ##1 }
6002         \tl_if_in:NnTF \l_tmpa_tl { - }
6003         { \@@_cut_on_hyphen:w ##1 \q_stop }
6004         { \def \l_tmpb_tl { ##1 } }
6005         \tl_if_empty:NTF \l_tmpa_tl
6006         { \def \l_tmpa_tl { 1 } }
6007         {
6008             \str_if_eq:eeT \l_tmpa_tl { * }
6009             { \def \l_tmpa_tl { 1 } }
6010         }
6011         \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
6012         { \@@_error:n { Invalid~col~number } }
6013         \tl_if_empty:NTF \l_tmpb_tl
6014         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6015         {
6016             \str_if_eq:eeT \l_tmpb_tl { * }
6017             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
6018         }
6019         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
6020         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

6021         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6022         \@@_qpoint:n { col - \l_tmpa_tl }
6023         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
6024         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }

```

```

6025     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6026     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } } }
6027     \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.

```

6028     \clist_map_inline:Nn \l_@@_rows_tl
6029     {
6030         \def \l_tmpa_tl { #####1 }
6031         \tl_if_in:NnTF \l_tmpa_tl { - }
6032         { \@@_cut_on_hyphen:w #####1 \q_stop }
6033         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
6034         \tl_if_empty:NnTF \l_tmpa_tl
6035         { \def \l_tmpa_tl { 1 } }
6036         {
6037             \str_if_eq:eeT \l_tmpa_tl { * }
6038             { \def \l_tmpa_tl { 1 } }
6039         }
6040         \tl_if_empty:NnTF \l_tmpb_tl
6041         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6042         {
6043             \str_if_eq:eeT \l_tmpb_tl { * }
6044             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
6045         }
6046         \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
6047         { \@@_error:n { Invalid~row~number } }
6048         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
6049         { \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`.

```

6050         \cs_if_exist:cF
6051         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
6052         {
6053             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
6054             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6055             \@@_qpoint:n { row - \l_tmpa_tl }
6056             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6057             \pgfpathrectanglecorners
6058             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6059             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6060         }
6061     }
6062 }
6063 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6064 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6065 {
6066     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6067     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6068     \clist_map_inline:Nn \l_@@_cols_tl
6069     {
6070         \@@_qpoint:n { col - ##1 }
6071         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
6072         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6073         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6074         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } } }
6075     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6076     \clist_map_inline:Nn \l_@@_rows_tl
6077     {
6078         \@@_if_in_corner:nF { #####1 - ##1 }

```

```

6079     {
6080         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6081         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6082         \@@_qpoint:n { row - #####1 }
6083         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6084         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6085         {
6086             \pgfpathrectanglecorners
6087             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6088             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6089         }
6090     }
6091 }
6092 }
6093 }

```

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`).

```

6094 \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

```

6095 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6096 {
6097     \bool_set_true:N \l_@@_nocolor_used_bool
6098     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6099     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6100     \clist_map_inline:Nn \l_@@_rows_tl
6101     {
6102         \clist_map_inline:Nn \l_@@_cols_tl
6103         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6104     }
6105 }

```

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.

```

6106 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6107 {
6108     \clist_set_eq:NN \l_tmpa_clist #1
6109     \clist_clear:N #1
6110     \clist_map_inline:Nn \l_tmpa_clist
6111     {

```

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

```

6112     \def \l_tmpa_tl { ##1 }
6113     \tl_if_in:NnTF \l_tmpa_tl { - }
6114     { \@@_cut_on_hyphen:w ##1 \q_stop }
6115     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6116     \bool_lazy_or:nnT
6117     { \str_if_eq_p:ee \l_tmpa_tl { * } }
6118     { \tl_if_blank_p:o \l_tmpa_tl }
6119     { \def \l_tmpa_tl { 1 } }
6120     \bool_lazy_or:nnT
6121     { \str_if_eq_p:ee \l_tmpb_tl { * } }
6122     { \tl_if_blank_p:o \l_tmpb_tl }
6123     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6124     \int_compare:nNnT \l_tmpb_tl > { #2 }
6125     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }

```

```

6126         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
6127         { \clist_put_right:Nn #1 { #####1 } }
6128     }
6129 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

6130 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6131 {
6132     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6133     {

```

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`).

```

6134         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6135         { \int_use:N \c@iRow - \int_use:N \c@jCol }
6136     }
6137     \ignorespaces
6138 }

6139 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6140 { \@@_error:n { cellcolor~in~Block } }
6141 % \end{macrocode}
6142 %
6143 % \begin{macrocode}
6144 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6145 { \@@_error:n { rowcolor~in~Block } }
6146 % \end{macrocode}
6147 %
6148 % \bigskip
6149 % The following command will be linked to |\rowcolor| in the tabular.
6150 % \begin{macrocode}
6151 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6152 {
6153     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6154     {
6155         \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6156         { \int_use:N \c@iRow - \int_use:N \c@jCol }
6157         { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6158     }
6159     \ignorespaces
6160 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

6161 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6162 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

6163 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6164 {

```

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`.

```

6165     \seq_gclear:N \g_tmpa_seq
6166     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6167     { \@@_rowlistcolors_tabular:nnnn #1 }
6168     \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`.

```

6169 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6170 {
6171   { \int_use:N \c@iRow }
6172   { \exp_not:n { #1 } }
6173   { \exp_not:n { #2 } }
6174   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6175 }
6176 \ignorespaces
6177 }

```

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`).

```

6178 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6179 {
6180   \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).

```

6181   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6182   {
6183     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6184     {
6185       \@@_rowlistcolors
6186       [ \exp_not:n { #2 } ]
6187       { #1 - \int_eval:n { \c@iRow - 1 } }
6188       { \exp_not:n { #3 } }
6189       [ \exp_not:n { #4 } ]
6190     }
6191   }
6192 }

```

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.

```

6193 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6194 {
6195   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6196   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6197   \seq_gclear:N \g_@@_rowlistcolors_seq
6198 }

6199 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6200 {
6201   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6202   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6203 }

```

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.

```

6204 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6205 {

```

With the following line, we test whether the cell is the first one that we actually encounter in its column (don't forget that some rows may be incomplete and that an instruction `\multicolumn` may be used, leading to cells which are not “evaluated”).

```
6206 \seq_if_in:NVF \g_@@_columncolor_seq \c@jCol
6207 {
6208 \seq_gput_right:NV \g_@@_columncolor_seq \c@jCol
```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```
6209 \tl_gput_left:Nc \g_@@_pre_code_before_tl
6210 {
6211 \exp_not:N \columncolor [ #1 ]
6212 { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6213 }
6214 }
6215 }

6216 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6217 {
6218 \clist_map_inline:nn { #1 }
6219 {
6220 \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6221 { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6222 \columncolor { nocolor } { ##1 }
6223 }
6224 }

6225 \cs_new_protected:Npn \@@_EmptyRow:n #1
6226 {
6227 \clist_map_inline:nn { #1 }
6228 {
6229 \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6230 { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6231 \rowcolor { nocolor } { ##1 }
6232 }
6233 }
```

## 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`.

```
6234 \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`.

```
6235 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6236 {
```



```

6237 \int_if_zero:nTF \l_@@_first_col_int
6238 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6239 {
6240   \int_if_zero:nTF \c@jCol
6241   {
6242     \int_compare:nNnF \c@iRow = { -1 }
6243     {
6244       \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 }
6245       { #1 }
6246     }
6247   }
6248   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6249 }
6250 }

```

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.

```

6251 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6252 {
6253   \int_if_zero:nF \c@iRow
6254   {
6255     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6256     {
6257       \int_compare:nNnT \c@jCol > \c_zero_int
6258       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6259     }
6260   }
6261 }

```

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`.

```

6262 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6263 {
6264   \IfPackageLoadedTF { tikz }
6265   {
6266     \IfPackageLoadedTF { booktabs }
6267     { #2 }
6268     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6269   }
6270   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6271 }
6272 \NewExpandableDocumentCommand { \@@_TopRule } { }
6273 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6274 \cs_new:Npn \@@_TopRule_i:
6275 {
6276   \noalign \bgroup
6277   \peek_meaning:NTF [
6278   { \@@_TopRule_ii: }
6279   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6280 }
6281 \NewDocumentCommand \@@_TopRule_ii: { o }
6282 {
6283   \tl_gput_right:Ne \g_@@_rules_tl
6284   {
6285     \@@_draw_hrule:n
6286     {
6287       position = \int_eval:n { \c@iRow + 1 } ,
6288       tikz =

```

```

6289         {
6290             line~width = #1 ,
6291             yshift = 0.25 \arrayrulewidth ,
6292             shorten~< = - 0.5 \arrayrulewidth
6293         } ,
6294         total~width = #1
6295     }
6296 }
6297 \skip_vertical:n { \belowrulesep + #1 }
6298 \egroup
6299 }
6300 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6301 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6302 \cs_new:Npn \@@_BottomRule_i:
6303 {
6304     \noalign \bgroup
6305     \peek_meaning:NTF [
6306         { \@@_BottomRule_ii: }
6307         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6308     }
6309 \NewDocumentCommand \@@_BottomRule_ii: { o }
6310 {
6311     \tl_gput_right:Ne \g_@@_rules_tl
6312     {
6313         \@@_draw_hrulerule:n
6314         {
6315             position = \int_eval:n { \c@iRow + 1 } ,
6316             tikz =
6317             {
6318                 line~width = #1 ,
6319                 yshift = 0.25 \arrayrulewidth ,
6320                 shorten~< = - 0.5 \arrayrulewidth
6321             } ,
6322             total~width = #1 ,
6323         }
6324     }
6325     \skip_vertical:N \aboverulesep
6326     \@@_create_row_node_i:
6327     \skip_vertical:n { #1 }
6328     \egroup
6329 }
6330 \NewExpandableDocumentCommand { \@@_MidRule } { }
6331 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6332 \cs_new:Npn \@@_MidRule_i:
6333 {
6334     \noalign \bgroup
6335     \peek_meaning:NTF [
6336         { \@@_MidRule_ii: }
6337         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6338     }
6339 \NewDocumentCommand \@@_MidRule_ii: { o }
6340 {
6341     \skip_vertical:N \aboverulesep
6342     \@@_create_row_node_i:
6343     \tl_gput_right:Ne \g_@@_rules_tl
6344     {
6345         \@@_draw_hrulerule:n
6346         {
6347             position = \int_eval:n { \c@iRow + 1 } ,
6348             tikz =
6349             {

```

```

6350         line-width = #1 ,
6351         yshift = 0.25 \arrayrulewidth ,
6352         shorten~< = - 0.5 \arrayrulewidth
6353     } ,
6354     total-width = #1 ,
6355 }
6356 }
6357 \skip_vertical:n { \belowrulesep + #1 }
6358 \egroup
6359 }

```

## General system for drawing rules

```

6360 \AtBeginDocument
6361 {
6362   \cs_new_protected:Npe \@@_draw_rules:
6363   {
6364     \c_@@_pgfortikzpicture_tl
6365     \@@_draw_rules_i:
6366     \c_@@_endpgfortikzpicture_tl
6367   }
6368 }
6369 \cs_new_protected:Npn \@@_draw_rules_i:
6370 {
6371   \bool_lazy_all:nT
6372   {
6373     { \clist_if_empty_p:N \l_@@_corners_clist }
6374     { \seq_if_empty_p:N \g_@@_pos_of_blocks_seq }
6375     { \seq_if_empty_p:N \g_@@_pos_of_xdots_seq }
6376     { \seq_if_empty_p:N \g_@@_pos_of_stroken_blocks_seq }
6377     { ! \l_@@_fix_vertex_bool } % added 2026/04/28
6378   }
6379   {
6380     \socket_assign_plug:nn { nicematrix / draw-hrule } { quick }
6381     \socket_assign_plug:nn { nicematrix / draw-vrule } { quick }
6382   }
6383   \pgfrememberpicturepositiononpagetrue
6384   \pgf@relevantforpicturesizefalse
6385   \pgfsetrectcap
6386   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6387   \CT@arc@
6388   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_key_hlines:
6389   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_key_vlines:
6390   \g_@@_rules_tl
6391 }

```

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”.

```

6392 \keys_define:nn { nicematrix / rules-after }
6393 {
6394   position .int_set:N = \l_@@_position_int ,
6395   start .int_set:N = \l_@@_start_int ,
6396   start .initial:n = 1 ,
6397   end .int_set:N = \l_@@_end_int ,
6398   multiplicity .code:n = \@@_set_multiplicity:n { #1 } ,
6399   dotted .code:n =
6400     \bool_set_true:N \l_@@_dotted_bool
6401     \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
6402     \socket_assign_plug:nn { nicematrix / hsegment } { dotted } ,
6403   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]`.

```

6404     color .code:n =
6405         \@@_set_CTarc:n { #1 }
6406         \CT@arc@
6407         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6408     color .value_required:n = true ,
6409     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6410     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
    }
}

6411     total-width .dim_set:N = \l_@@_rule_width_after_dim ,
6412     unknown .code:n =
6413         \@@_unknown_key:nn
6414         { nicematrix / rules-after }
6415         { Unknown-key-for-a-rule } ,
6416     tikz .value_required:n = true
6417 }

```

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.

```

6418 \AtBeginDocument
6419 {
6420     \IfPackageLoadedTF { tikz }
6421     {
6422         \keys_define:nn { nicematrix / rules-after }
6423         {
6424             tikz .code:n =
6425                 \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 }
6426                 \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
6427                 \socket_assign_plug:nn { nicematrix / hsegment } { tikz } ,
6428             dashed .meta:n =
6429                 {
6430                     tikz = nicematrix / dashed ,
6431                     total-width = \pgflinewidth
6432                 }
6433         }
6434     }
6435     {
6436         \keys_define:nn { nicematrix / rules-after }
6437         { tikz .code:n = \@@_error:n { tikz-without-tikz } }
6438     }
6439 }

6440 \cs_new_protected:Npn \@@_set_multiplicity:n #1
6441 {
6442     \int_set:Nn \l_@@_multiplicity_int { #1 }
6443     \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
6444     \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
6445 }

```

```

6446 \AtBeginDocument
6447 {
6448   \IfPackageLoadedT { tikz }
6449   {
6450     \tikzset
6451     {
6452       nicematrix / dashed / .style =
6453       {
6454         dash-pattern = on~4~pt~off~2pt ,
6455         line-cap = butt
6456       }
6457     }
6458     \cs_if_exist:cT { tikz@library@decorations@loaded }
6459     { \tikzset { nicematrix / dashed / .append-style = dash-expand-off } }
6460   }
6461 }

```

### 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.

```

6462 \cs_new_protected:Npn \@@_draw_vrule:n #1
6463 {

```

The group is for the options.

```

6464 {
6465   \int_set_eq:NN \l_@@_end_int \c@iRow
6466   \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).

```

6467   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6468   \socket_use:n { nicematrix / draw-vrule }
6469 }
6470 }

```

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.

```

6471 \socket_new:nn { nicematrix / draw-vrule } { 0 }
6472 \socket_new_plug:nnn { nicematrix / draw-vrule } { general }
6473 {
6474   \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`.

```

6475   \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).

```

6476 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6477 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6478 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6479 \l_tmpa_tl
6480 {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6481 \@@_test_v:
6482 \bool_if:NTF \g_tmpa_bool
6483 {
6484 \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.

```

6485 { \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl }
6486 { \socket_use:nn { nicematrix / draw-at-odd-vertex-v } }
6487 }
6488 {
6489 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6490 {
6491 \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }

```

The socket `vsegment` is defined below with its four plugs.

```

6492 \socket_use:n { nicematrix / vsegment }
6493 \int_zero:N \l_@@_segment_start_int
6494 }
6495 }
6496 }
6497 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6498 {
6499 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6500 \socket_use:n { nicematrix / vsegment }
6501 }
6502 \group_end:
6503 }

6504 \socket_assign_plug:nn { nicematrix / draw-vrule } { general }
6505 \socket_new_plug:nnn { nicematrix / draw-vrule } { quick }
6506 {
6507 \group_begin:
6508 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6509 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6510 \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6511 \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6512 \socket_use:n { nicematrix / vsegment }
6513 \group_end:
6514 }

```

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.

```

6515 \cs_new_protected:Npn \@@_test_v:
6516 {
6517 \bool_gset_true:N \g_tmpa_bool
6518 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6519 \bool_if:NT \g_tmpa_bool
6520 {
6521 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6522 { \@@_test_vline_in_block:nnnnn ##1 }
6523 \bool_if:NT \g_tmpa_bool
6524 {
6525 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq

```

```

6526         { \@@_test_vline_in_block:nnnnn ##1 }
6527     \bool_if:NT \g_tmpa_bool
6528     {
6529         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6530         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6531     }
6532 }
6533 }
6534 }
6535 \socket_new:nn { nicematrix / draw-at-odd-vertex-v } { 0 }
6536 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-v } { active }
6537 {
6538     {
6539         \int_compare:nNnTF \l_@@_position_int > \c@jCol
6540         { \bool_set_false:N \l_tmpa_bool }
6541         {
6542             \@@_test_h:
6543             \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6544         }
6545         \int_compare:nNnTF \l_@@_position_int = 1
6546         { \bool_gset_false:N \g_tmpa_bool }
6547         {
6548             \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl - 1 } }
6549             \@@_test_h:
6550         }
6551         \bool_gset:Nn \g_tmpa_bool
6552         { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6553     }
6554     \bool_if:NT \g_tmpa_bool
6555     {
6556         \int_set:Nn \l_@@_segment_end_int { \l_tmpa_tl - 1 }
6557         \socket_use:n { nicematrix / vsegment }
6558         \int_set:Nn \l_@@_segment_start_int \l_tmpa_tl
6559     }
6560 }
6561 \cs_new_protected:Npn \@@_test_in_corner_v:
6562 {
6563     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6564     {
6565         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6566         { \bool_set_false:N \g_tmpa_bool }
6567     }
6568     {
6569         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6570         {
6571             \int_compare:nNnTF \l_tmpb_tl = 1
6572             { \bool_set_false:N \g_tmpa_bool }
6573             {
6574                 \@@_if_in_corner:nT
6575                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6576                 { \bool_set_false:N \g_tmpa_bool }
6577             }
6578         }
6579     }
6580 }

```

For the drawing of the (vertical) segments, we will use a socket with four plugs :

- a plug `standard` for the simple rules.
- a plug `multiple` for the rules similar to the standard rules of `array` and `colortbl`, that is to say with `multiplicity`, etc;

- a plug dotted for the rules with our own system of dotted rules;
- a plug tikz for the rules drawn with TikZ, including the key dashed of custom-line.

```
6581 \socket_new:nn { nicematrix / vsegment } { 0 }
```

In the following plug, the  $x$ -value for the line has been computed previously in  $\backslash l\_@@\_x\_initial\_dim$ .

```
6582 \socket_new_plug:nnn { nicematrix / vsegment } { standard }
6583 {
6584   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6585   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6586   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6587   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \pgf@y }
6588   \pgfusepathqstroke
6589 }

6590 \socket_assign_plug:nn { nicematrix / vsegment } { standard }
6591 \socket_new_plug:nnn { nicematrix / vsegment } { multiple }
6592 {
6593   \dim_add:Nn \l_@@_x_initial_dim
6594   {
6595     - 0.5 \l_@@_rule_width_after_dim
6596     +
6597     ( \arrayrulewidth * \l_@@_multiplicity_int
6598       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6599   }
6600   \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6601   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6602   \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6603   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6604   \bool_lazy_and:nnT
6605   { \cs_if_exist_p:N \CT@drsc@ }
6606   { ! \tl_if_blank_p:o \CT@drsc@ }
6607   {
6608     {
6609       \CT@drsc@
6610       \dim_add:Nn \l_@@_y_initial_dim { 0.5 \arrayrulewidth }
6611       \dim_sub:Nn \l_@@_y_final_dim { 0.5 \arrayrulewidth }
6612       \dim_set:Nn \l_@@_tmpd_dim
6613       {
6614         \l_@@_x_initial_dim - ( \doublerulesep + \arrayrulewidth )
6615         * ( \l_@@_multiplicity_int - 1 )
6616       }
6617       \pgfpathrectanglecorners
6618       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6619       { \pgfpoint \l_@@_tmpd_dim \l_@@_y_final_dim }
6620       \pgfusepath { fill }
6621     }
6622   }
6623   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6624   \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6625   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6626   {
6627     \dim_sub:Nn \l_@@_x_initial_dim { \arrayrulewidth + \doublerulesep }
6628     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6629     \pgfpathlineto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_final_dim }
6630   }
6631   \pgfusepathqstroke
6632 }

6633 \socket_new_plug:nnn { nicematrix / vsegment } { dotted }
6634 {
6635   \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
```



```

6636 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6637 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6638 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6639 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }
6640 \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`

```

6641 \@@_draw_line:
6642 }

```

```

6643 \socket_new_plug:nnn { nicematrix / vsegment } { tikz }
6644 {
6645 {

```

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).

```

6646 \tl_if_empty:NF \l_@@_rule_color_tl
6647 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6648 \@@_qpoint:n { row - \int_use:N \l_@@_segment_start_int }
6649 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6650 \dim_sub:Nn \l_@@_x_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6651 \@@_qpoint:n { row - \int_eval:n { \l_@@_segment_end_int + 1 } }

```

The following line can't be short-cutted.

```

6652 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6653 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6654 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim ) --
6655 ( \l_@@_x_initial_dim , \l_@@_y_final_dim ) ;
6656 }
6657 }

```

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).

```

6658 \cs_new_protected:Npn \@@_draw_key_vlines:
6659 {
6660 {
6661 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6662 \int_set:Nn \l_@@_end_int \c@iRow

```

There is a currying in the following code.

```

6663 \str_if_eq:eeTF \l_@@_vlines_clist { all }
6664 { \@@_lines_step_inline:n \c@jCol }
6665 { \clist_map_inline:Nn \l_@@_vlines_clist }
6666 {
6667 \int_set:Nn \l_@@_position_int { ##1 }
6668 \socket_use:n { nicematrix / draw-vrule }
6669 }
6670 }
6671 }

```

The following command is used in `\@@_draw_key_vlines:` and in `\@@_draw_key_hlines:`.

```

6672 \cs_new_protected:Npn \@@_lines_step_inline:n #1
6673 {
6674   \int_step_inline:nnn
6675   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6676   {
6677     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6678     { #1 }
6679     { \int_eval:n { #1 + 1 } }
6680   }
6681 }

```

## The horizontal rules

`\@@_draw_hrule:n` and the socket `draw-hrule` will appear in `\@@_draw_key_hlines:n` and in the token rules `\g_@@_rules_tl` (or within other functions used in `\g_@@_rules_tl`) and those token lists will be executed within a PGF pseudo-environment.

The argument `#1` is a list of *key=value* pairs.

```

6682 \cs_new_protected:Npn \@@_draw_hrule:n #1
6683 {
6684   {
6685     \int_set_eq:NN \l_@@_end_int \c@jCol
6686     \keys_set_known:nn { nicematrix / rules-after } { #1 }
6687     \socket_use:nn { nicematrix / draw-hrule }
6688   }
6689 }

```

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.

```

6690 \socket_new:nn { nicematrix / draw-hrule } { 0 }
6691 \socket_new_plug:nnn { nicematrix / draw-hrule } { general }
6692 {
6693   \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`.

```

6694   \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).

```

6695   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6696   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6697   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6698   \l_tmpb_tl
6699   {

```

After the execution of `\test_v:`, `\g_tmpa_bool` will be equal to `true` if we have to draw that rule.

```

6700   \@@_test_h:
6701   \bool_if:NTF \g_tmpa_bool
6702   {
6703     \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.

```

6704         { \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl }
6705         { \socket_use:n { nicematrix / draw-at-odd-vertex-h } }
6706     }
6707     {
6708         \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6709         {
6710             \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }

```

The socket `hsegment` is defined below with its four plugs.

```

6711         \socket_use:n { nicematrix / hsegment }
6712         \int_zero:N \l_@@_segment_start_int
6713     }
6714 }
6715 }
6716 \int_compare:nNnT \l_@@_segment_start_int > \c_zero_int
6717 {
6718     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6719     \socket_use:n { nicematrix / hsegment }
6720 }
6721 \group_end:
6722 }
6723 \socket_assign_plug:nn { nicematrix / draw-hrule } { general }
6724 \socket_new_plug:nnn { nicematrix / draw-hrule } { quick }
6725 {
6726     \group_begin:
6727     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6728     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6729     \int_set_eq:NN \l_@@_segment_start_int \l_@@_start_int
6730     \int_set_eq:NN \l_@@_segment_end_int \l_@@_end_int
6731     \socket_use:n { nicematrix / hsegment }
6732     \group_end:
6733 }

```

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.

```

6734 \cs_new_protected:Npn \@@_test_h:
6735 {
6736     \bool_gset_true:N \g_tmpa_bool
6737     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6738     \bool_if:NT \g_tmpa_bool
6739     {
6740         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6741         { \@@_test_hline_in_block:nnnnn ##1 }
6742         \bool_if:NT \g_tmpa_bool
6743         {
6744             \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6745             { \@@_test_hline_in_block:nnnnn ##1 }
6746             \bool_if:NT \g_tmpa_bool
6747             {
6748                 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6749                 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6750             }
6751         }
6752     }
6753 }
6754 \socket_new:nn { nicematrix / draw-at-odd-vertex-h } { 0 }

```

```

6755 \socket_new_plug:nnn { nicematrix / draw-at-odd-vertex-h } { active }
6756 {
6757   {
6758     \int_compare:nNnTF \l_@@_position_int > \c@iRow
6759     { \bool_set_false:N \l_tmpa_bool }
6760     {
6761       \@@_test_v:
6762       \bool_set_eq:NN \l_tmpa_bool \g_tmpa_bool
6763     }
6764     \int_compare:nNnTF \l_@@_position_int = 1
6765     { \bool_gset_false:N \g_tmpa_bool }
6766     {
6767       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl - 1 } }
6768       \@@_test_v:
6769     }
6770     \bool_gset:Nn \g_tmpa_bool
6771     { \bool_xor_p:nn \g_tmpa_bool \l_tmpa_bool }
6772   }
6773   \bool_if:NT \g_tmpa_bool
6774   {
6775     \int_set:Nn \l_@@_segment_end_int { \l_tmpb_tl - 1 }
6776     \socket_use:n { nicematrix / hsegment }
6777     \int_set:Nn \l_@@_segment_start_int \l_tmpb_tl
6778   }
6779 }

6780 \cs_new_protected:Npn \@@_test_in_corner_h:
6781 {
6782   \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6783   {
6784     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6785     { \bool_set_false:N \g_tmpa_bool }
6786   }
6787   {
6788     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6789     {
6790       \int_compare:nNnTF \l_tmpa_tl = 1
6791       { \bool_set_false:N \g_tmpa_bool }
6792       {
6793         \@@_if_in_corner:nT
6794         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6795         { \bool_set_false:N \g_tmpa_bool }
6796       }
6797     }
6798   }
6799 }

```

For the drawing of the (horizontal) segments, we will use a socket with four plugs :

- a plug **standard** for simple rules;
- a plug **multiple** for the rules similar to the standard rules of `array` and `colortbl`, that is to say with multiplicity, etc;
- a plug **dotted** for the rules with our own system of dotted rules;
- a plug **tikz** for the rules drawn with TikZ, including the key `dashed` of `custom-line`.

```

6800 \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`.

```

6801 \socket_new_plug:nnn { nicematrix / hsegment } { standard }

```

```

6802 {
6803   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6804   \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6805   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6806   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6807   \pgfusepathqstroke
6808 }
6809 \socket_assign_plug:nn { nicematrix / hsegment } { standard }
6810 \socket_new_plug:nnn { nicematrix / hsegment } { multiple }
6811 {
6812   \dim_add:Nn \l_@@_y_initial_dim
6813   {
6814     - 0.5 \l_@@_rule_width_after_dim
6815     +
6816     ( \arrayrulewidth * \l_@@_multiplicity_int
6817       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6818   }
6819   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6820   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6821   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6822   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6823   \bool_lazy_and:nnT
6824   { \cs_if_exist_p:N \CT@drsc@ }
6825   { ! \tl_if_blank_p:o \CT@drsc@ }
6826   {
6827     {
6828       \CT@drsc@
6829       \dim_set:Nn \l_@@_tmpd_dim
6830       {
6831         \l_@@_y_initial_dim - ( \doublerulesep + \arrayrulewidth )
6832         * ( \l_@@_multiplicity_int - 1 )
6833       }
6834       \pgfpathrectanglecorners
6835       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6836       { \pgfpoint \l_@@_x_final_dim \l_@@_tmpd_dim }
6837       \pgfusepathqfill
6838     }
6839   }
6840   \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6841   \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6842   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6843   {
6844     \dim_sub:Nn \l_@@_y_initial_dim { \arrayrulewidth + \doublerulesep }
6845     \pgfpathmoveto { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
6846     \pgfpathlineto { \pgfpoint \l_@@_x_final_dim \l_@@_y_initial_dim }
6847   }
6848   \pgfusepathqstroke
6849 }

```

Now, the case of a dotted line.

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{array}{|cccc|}
 \hline
 1 & 2 & 3 & 4 \\
 1 & 2 & 3 & 4 \\
 1 & 2 & 3 & 4 \\
 \hline
 \end{array}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6850 \socket_new_plug:nnn { nicematrix / hsegment } { dotted }
6851 {
6852   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }
6853   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6854   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6855   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6856   \int_compare:nNnT \l_@@_segment_start_int = 1
6857   {
6858     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6859     \bool_if:NF \g_@@_delims_bool
6860     { \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.

```

6861   \tl_if_eq:NnF \g_@@_left_delim_tl (
6862     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6863   )
6864   \@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6865   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6866   \int_compare:nNnT \l_@@_segment_end_int = \c@jCol
6867   {
6868     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6869     \bool_if:NF \g_@@_delims_bool
6870     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6871     \tl_if_eq:NnF \g_@@_right_delim_tl )
6872     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6873   }

```

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`

```

6874   \@@_draw_line:
6875 }

6876 \socket_new_plug:nnn { nicematrix / hsegment } { tikz }
6877 {
6878   {

```

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).

```

6879   \tl_if_empty:NF \l_@@_rule_color_tl
6880   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6881   \@@_qpoint:n { col - \int_use:N \l_@@_segment_start_int }
6882   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6883   \dim_sub:Nn \l_@@_y_initial_dim { 0.5 \l_@@_rule_width_after_dim }

```

```

6884 \l_@@_qpoint:n { col - \int_eval:n { \l_@@_segment_end_int + 1 } }
6885 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6886 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6887 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
6888 -- ( \l_@@_x_final_dim , \l_@@_y_initial_dim ) ;
6889 }
6890 }

```

The command `\l_@@_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).

```

6891 \cs_new_protected:Npn \l_@@_draw_key_hlines:
6892 {
6893 {
6894 \dim_set_eq:NN \l_@@_rule_width_after_dim \arrayrulewidth
6895 \int_set:Nn \l_@@_end_int \c@jCol

```

There is a currying in the following code.

```

6896 \str_if_eq:eeTF \l_@@_hlines_clist { all }
6897 { \l_@@_lines_step_inline:n \c@iRow }
6898 { \clist_map_inline:Nn \l_@@_hlines_clist }
6899 {
6900 \int_set:Nn \l_@@_position_int { ##1 }
6901 \socket_use:n { nicematrix / draw-hrule }
6902 }
6903 }
6904 }

```

The command `\l_@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6905 \cs_set:Npn \l_@@_Hline: { \noalign \bgroup \l_@@_Hline_i:n { 1 } }

```

The argument of the command `\l_@@_Hline_i:n` is the number of successive `\Hline` found.

```

6906 \cs_set:Npn \l_@@_Hline_i:n #1
6907 {
6908 \peek_remove_spaces:n
6909 {
6910 \peek_meaning:NNTF \Hline
6911 { \l_@@_Hline_ii:nn { #1 + 1 } }
6912 { \l_@@_Hline_iii:n { #1 } }
6913 }
6914 }

6915 \cs_set:Npn \l_@@_Hline_ii:nn #1 #2 { \l_@@_Hline_i:n { #1 } }

6916 \cs_set:Npn \l_@@_Hline_iii:n #1
6917 { \l_@@_collect_options:n { \l_@@_Hline_iv:nn { #1 } } }

6918 \cs_set_protected:Npn \l_@@_Hline_iv:nn #1 #2
6919 {
6920 \l_@@_compute_rule_width:n { multiplicity = #1 , #2 }
6921 \skip_vertical:N \l_@@_rule_width_before_dim
6922 \tl_gput_right:Ne \g_@@_rules_tl
6923 {

```

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

```

\l_@@_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:

```

6924     {
6925         \int_compare:nNt { #1 } > 1 { \@@_set_multiplicity:n { #1 } }
6926         \int_set:Nn \l_@@_position_int { \int_eval:n { \c@iRow + 1 } }
6927         \dim_set:Nn \l_@@_rule_width_after_dim
6928             { \dim_use:N \l_@@_rule_width_before_dim }
6929         \@@_draw_hrule:n { #2 }
6930     }
6931 }
6932 \egroup
6933 }

```

### 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`).

```

6934 \cs_new_protected:Npn \@@_custom_line:n #1
6935 {
6936     \str_clear:N \l_@@_letter_str
6937     \str_clear:N \l_@@_command_str
6938     \str_clear:N \l_@@_ccommand_str
6939     \tl_clear_new:N \l_@@_other_keys_tl
6940     \keys_set_known:nn { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6941     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6942     {
6943         \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6944     \str_set:Ne \l_@@_command_str
6945         { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6946     }
6947     \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6948     {
6949         \str_set:Ne \l_@@_ccommand_str
6950             { \str_tail:N \l_@@_ccommand_str }
6951         \str_set:Ne \l_@@_ccommand_str
6952             { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6953     }

```

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.

```

6954     \bool_lazy_all:nTF
6955     {
6956         { \str_if_empty_p:N \l_@@_letter_str }
6957         { \str_if_empty_p:N \l_@@_command_str }
6958         { \str_if_empty_p:N \l_@@_ccommand_str }
6959     }
6960     { \@@_error:n { No~letter~and~no~command } }
6961     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6962 }
6963 \keys_define:nn { nicematrix / custom-line }
6964 {
6965     letter .str_set:N = \l_@@_letter_str ,
6966     letter .value_required:n = true ,
6967     command .str_set:N = \l_@@_command_str ,
6968     command .value_required:n = true ,
6969     ccommand .str_set:N = \l_@@_ccommand_str ,
6970     ccommand .value_required:n = true
6971 }

```



```

6972 \cs_new_protected:Npn \@@_custom_line_i:n #1
6973 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6974 \bool_set_false:N \l_@@_tikz_rule_bool
6975 \bool_set_false:N \l_@@_dotted_rule_bool
6976 \bool_set_false:N \l_@@_color_bool
6977 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6978 \bool_if:NT \l_@@_tikz_rule_bool
6979 {
6980   \IfPackageLoadedF { tikz }
6981     { \@@_error:n { tikz~in~custom-line-without~tikz } }
6982   \bool_if:NT \l_@@_color_bool
6983     { \@@_error:n { color~in~custom-line-with~tikz } }
6984 }
6985 \bool_if:NT \l_@@_dotted_rule_bool
6986 {
6987   \int_compare:nNnT \l_@@_multiplicity_int > 1
6988     { \@@_error:n { key~multiplicity~with~dotted } }
6989 }
6990 \str_if_empty:NF \l_@@_letter_str
6991 {
6992   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6993     { \@@_error:n { Several~letters } }
6994     {
6995       \tl_if_in:NoTF
6996         \c_@@_forbidden_letters_str
6997         \l_@@_letter_str
6998         { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6999     }

```

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.

```

7000           \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
7001           { \@@_v_custom_line:nn { #1 } }
7002       }
7003   }
7004 }
7005 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
7006 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
7007 }
7008 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
7009 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
7010 \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.

```

7011 \keys_define:nn { nicematrix / custom-line-bis }
7012 {
7013   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7014   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7015   color .value_required:n = true ,
7016   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7017   tikz .value_required:n = true ,
7018   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7019   dotted .value_forbidden:n = true ,
7020   dashed .meta:n =
7021   {

```

```

7022         tikz = nicematrix / dashed ,
7023         total-width = \pgflinewidth
7024     } ,
7025     total-width .code:n = { } ,
7026     total-width .value_required:n = true ,
7027     sep-color .code:n = { } ,
7028     sep-color .value_required:n = true ,
7029     unknown .code:n =
7030         \@@_unknown_key:nn
7031         { nicematrix / custom-line-bis }
7032         { Unknown-key-for-custom-line }
7033 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

7034 \bool_new:N \l_@@_dotted_rule_bool
7035 \bool_new:N \l_@@_tikz_rule_bool
7036 \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).

```

7037 \keys_define:nn { nicematrix / custom-line-width }
7038 {
7039     multiplicity .int_set:N = \l_@@_tmpc_int ,
7040     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
7041     total-width .code:n = \dim_set:Nn \l_@@_rule_width_before_dim { #1 }
7042         \bool_set_true:N \l_@@_total_width_bool ,
7043     total-width .value_required:n = true ,
7044     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
7045 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_draw_hrrule:n` (which is in the internal `\CodeAfter`).

```

7046 \cs_new_protected:Npn \@@_h_custom_line:n #1
7047 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign` (which is in `\Hline`).

```

7048     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
7049     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
7050 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_draw_hrrule:n` (which is in the internal `\CodeAfter`).

```

7051 \cs_new_protected:Npn \@@_c_custom_line:n #1
7052 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

7053     \exp_args:Nc \DeclareExpandableDocumentCommand
7054     { nicematrix - \l_@@_ccommand_str }
7055     { 0 { } m }
7056     {
7057         \noalign
7058         {
7059             \@@_compute_rule_width:n { #1 , ##1 }
7060             \skip_vertical:n \l_@@_rule_width_before_dim
7061             \clist_map_inline:nn
7062             { ##2 }

```

```

7063         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
7064     }
7065 }
7066 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
7067 }

```

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*.

```

7068 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
7069 {
7070     \tl_if_in:nnTF { #2 } { - }
7071     { \@@_cut_on_hyphen:w #2 \q_stop }
7072     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
7073     \tl_gput_right:Ne \g_@@_rules_tl
7074     {
7075         \@@_draw_hrule:n
7076         {
7077             #1 ,
7078             start = \l_tmpa_tl ,
7079             end = \l_tmpb_tl ,
7080             position = \int_eval:n { \c@iRow + 1 } ,
7081             total-width = \dim_use:N \l_@@_rule_width_before_dim
7082         }
7083     }
7084 }
7085 \cs_new_protected:Npn \@@_compute_rule_width:n #1
7086 {
7087     \bool_set_false:N \l_@@_tikz_rule_bool
7088     \bool_set_false:N \l_@@_total_width_bool
7089     \bool_set_false:N \l_@@_dotted_rule_bool
7090     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
7091     \bool_if:NF \l_@@_total_width_bool
7092     {
7093         \bool_if:NTF \l_@@_dotted_rule_bool
7094         { \dim_set:Nn \l_@@_rule_width_before_dim { 2 \l_@@_xdots_radius_dim } }
7095         {
7096             \bool_if:NF \l_@@_tikz_rule_bool
7097             {
7098                 \dim_set:Nn \l_@@_rule_width_before_dim
7099                 {
7100                     \arrayrulewidth * \l_@@_tmpc_int
7101                     + \doublerulesep * ( \l_@@_tmpc_int - 1 )
7102                 }
7103             }
7104         }
7105     }
7106 }

```

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

```

7107 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
7108 {
7109     \str_if_eq:nnTF { #2 } { [ ] }
7110     { \@@_v_custom_line_i:nw { #1 } [ ] }
7111     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
7112 }
7113 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
7114 { \@@_v_custom_line:nn { #1 , #2 } }
7115 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
7116 {
7117     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

7118 \tl_gput_right:Ne \g_@@_array_preamble_tl
7119 {
7120   \exp_not:N !
7121   { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_before_dim } }
7122 }
7123 \tl_gput_right:Ne \g_@@_rules_tl
7124 {

```

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.

```

7125 {
7126   \dim_set:Nn \l_@@_rule_width_after_dim
7127   { \dim_use:N \l_@@_rule_width_before_dim }
7128   \int_set:Nn \l_@@_position_int { \int_eval:n { \c@jCol + 1 } }
7129   \@@_draw_vrule:n { #2 }
7130 }
7131 }
7132 \@@_rec_preamble:n #1
7133 }
7134 \@@_custom_line:n
7135 { letter = : , command = \hdottedline , ccommand = \cdottedline, dotted }

```

## The key default-line

```

7136 \keys_define:nn { nicematrix / default-line }
7137 {
7138   multiplicity .int_set:N = \l_@@_multiplicity_int ,
7139   color .code:n = \bool_set_true:N \l_@@_color_bool ,
7140   color .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 \AtBeginDocument
7153 {
7154   \IfPackageLoadedT { tikz }
7155   {
7156     \keys_define:nn { nicematrix / default-line }
7157     {
7158       tikz .code:n =
7159         \bool_set_true:N \l_@@_tikz_rule_bool
7160         \tl_set:Nn \l_@@_tikz_rule_tl { #1 } ,
7161       tikz .value_required:n = true ,
7162       dashed .meta:n =
7163         {
7164           tikz = nicematrix / dashed ,
7165           total-width = \pgflinewidth
7166         }

```

```

7167     }
7168     \@@_custom_line:n
7169     {
7170         letter = ; ,
7171         command = \hdashedline ,
7172         ccommand = \cdashedline ,
7173         tikz = nicematrix / dashed ,
7174         total-width = \pgflinewidth
7175     }
7176 }
7177 }
7178 \cs_new_protected:Npn \@@_default_line:n #1
7179 {
7180     \bool_set_false:N \l_@@_tikz_rule_bool
7181     \bool_set_false:N \l_@@_dotted_rule_bool
7182     \bool_set_false:N \l_@@_color_bool
7183     \keys_set:nn { nicematrix / default-line } { #1 }
7184     \bool_if:NT \l_@@_tikz_rule_bool
7185     {
7186         \IfPackageLoadedF { tikz }
7187         { \@@_error:n { tikz~in~custom~line~without~tikz } }
7188         \bool_if:NT \l_@@_color_bool
7189         { \@@_error:n { color~in~custom~line~with~tikz } }
7190     }
7191     \bool_if:NT \l_@@_dotted_rule_bool
7192     {
7193         \int_compare:nNnT \l_@@_multiplicity_int > 1
7194         { \@@_error:n { key~multiplicity~with~dotted } }
7195     }
7196     \bool_if:NTF \l_@@_tikz_rule_bool
7197     {
7198         \socket_assign_plug:nn { nicematrix / vsegment } { tikz }
7199         \socket_assign_plug:nn { nicematrix / hsegment } { tikz }
7200     }
7201     {
7202         \bool_if:NTF \l_@@_dotted_rule_bool
7203         {
7204             \socket_assign_plug:nn { nicematrix / vsegment } { dotted }
7205             \socket_assign_plug:nn { nicematrix / hsegment } { dotted }
7206         }
7207         {
7208             \bool_if:NTF \l_@@_multiple_rule_bool
7209             {
7210                 \socket_assign_plug:nn { nicematrix / vsegment } { multiple }
7211                 \socket_assign_plug:nn { nicematrix / hsegment } { multiple }
7212             }
7213         }
7214     }
7215 }

```

## 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.

```

7216 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
7217 {
7218     \int_compare:nNnT \l_tmpa_tl > { #1 }
7219     {
7220         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7221         {
7222             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7223             {

```

```

7224         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7225         { \bool_gset_false:N \g_tmpa_bool }
7226     }
7227 }
7228 }
7229 }

```

The same for vertical rules.

```

7230 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
7231 {
7232     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7233     {
7234         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7235         {
7236             \int_compare:nNnT \l_tmpb_tl > { #2 }
7237             {
7238                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7239                 { \bool_gset_false:N \g_tmpa_bool }
7240             }
7241         }
7242     }
7243 }

7244 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
7245 {
7246     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
7247     {
7248         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
7249         {
7250             \int_compare:nNnTF \l_tmpa_tl = { #1 }
7251             { \bool_gset_false:N \g_tmpa_bool }
7252             {
7253                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
7254                 { \bool_gset_false:N \g_tmpa_bool }
7255             }
7256         }
7257     }
7258 }

7259 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
7260 {
7261     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
7262     {
7263         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
7264         {
7265             \int_compare:nNnTF \l_tmpb_tl = { #2 }
7266             { \bool_gset_false:N \g_tmpa_bool }
7267             {
7268                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
7269                 { \bool_gset_false:N \g_tmpa_bool }
7270             }
7271         }
7272     }
7273 }

```

## 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.

```

7274 \cs_new_protected:Npn \@@_compute_corners:
7275 {
7276   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7277   { \@@_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`).

```

7278   \clist_clear:N \l_@@_corners_cells_clist
7279   \clist_map_inline:Nn \l_@@_corners_clist
7280   {
7281     \str_case:nnF { ##1 }
7282     {
7283       { NW }
7284       { \@@_compute_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7285       { NE }
7286       { \@@_compute_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7287       { SW }
7288       { \@@_compute_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7289       { SE }
7290       { \@@_compute_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7291     }
7292     { \@@_error:nn { bad~corner } { ##1 } }
7293   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7294   \clist_if_empty:NF \l_@@_corners_cells_clist
7295   {

```

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.

```

7296     \tl_gput_right:Ne \g_@@_aux_tl
7297     {
7298       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7299       { \l_@@_corners_cells_clist }
7300     }
7301   }
7302 }

```

```

7303 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7304 {
7305   \int_step_inline:nnn { #1 } { #3 }
7306   {
7307     \int_step_inline:nnn { #2 } { #4 }
7308     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
7309   }
7310 }

```

```

7311 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7312 {
7313   \cs_if_exist:cTF
7314   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7315   \prg_return_true:
7316   \prg_return_false:
7317 }

```

“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.

```
7318 \cs_new_protected:Npn \@@_compute_corner:nnnnnn #1 #2 #3 #4 #5 #6
7319 {
```

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.

```
7320 \bool_set_false:N \l_tmpa_bool
7321 \int_zero_new:N \l_@@_last_empty_row_int
7322 \int_set:Nn \l_@@_last_empty_row_int { #1 }
7323 \int_step_inline:nnnn { #1 } { #3 } { #5 }
7324 {
7325   \bool_lazy_or:nnTF
7326   {
7327     \cs_if_exist_p:c
7328     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7329   }
7330   { \@@_if_in_block_p:nn { ##1 } { #2 } }
7331   { \bool_set_true:N \l_tmpa_bool }
7332   {
7333     \bool_if:NF \l_tmpa_bool
7334     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7335   }
7336 }
```

Now, you determine the last empty cell in the row of number 1.

```
7337 \bool_set_false:N \l_tmpa_bool
7338 \int_zero_new:N \l_@@_last_empty_column_int
7339 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7340 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7341 {
7342   \bool_lazy_or:nnTF
7343   {
7344     \cs_if_exist_p:c
7345     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7346   }
7347   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7348   { \bool_set_true:N \l_tmpa_bool }
7349   {
7350     \bool_if:NF \l_tmpa_bool
7351     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7352   }
7353 }
```

Now, we loop over the rows.

```
7354 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
7355 {
```

We treat the row number `##1` with another loop.

```
7356 \bool_set_false:N \l_tmpa_bool
7357 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
7358 {
7359   \bool_lazy_or:nnTF
7360   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7361   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7362   { \bool_set_true:N \l_tmpa_bool }
7363   {
7364     \bool_if:NF \l_tmpa_bool
```



```

7365         {
7366             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7367             \clist_put_right:Nn
7368                 \l_@@_corners_cells_clist
7369                 { ##1 - #####1 }
7370             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7371         }
7372     }
7373 }
7374 }
7375 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7376 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7377 \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:NcT \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.

```

7378 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

7379 \keys_define:nn { nicematrix / NiceMatrixBlock }
7380 {
7381     auto-columns-width .code:n =
7382     {
7383         \bool_set_true:N \l_@@_block_auto_columns_width_bool
7384         \dim_gzero_new:N \g_@@_max_cell_width_dim
7385         \bool_set_true:N \l_@@_auto_columns_width_bool
7386     }
7387 }

7388 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
7389 {
7390     \int_gincr:N \g_@@_NiceMatrixBlock_int
7391     \dim_zero:N \l_@@_columns_width_dim
7392     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7393     \bool_if:NT \l_@@_block_auto_columns_width_bool
7394     {
7395         \cs_if_exist:cT
7396             { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7397         {
7398             \dim_set:Nn \l_@@_columns_width_dim
7399             {
7400                 \use:c
7401                     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7402             }
7403         }
7404     }
7405 }

```

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`).

```
7406 {
7407   \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.

```
7408   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7409   {
7410     \bool_if:NT \l_@@_block_auto_columns_width_bool
7411     {
7412       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7413       \iow_shipout:Ne \@mainaux
7414       {
7415         \cs_gset:cpn
7416         { @@ _ 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.

```
7417         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7418       }
7419       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7420     }
7421   }
7422   \ignorespacesafterend
7423 }
```

## 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).

```
7424 \cs_new_protected:Npn \@@_create_extra_nodes:
7425 {
7426   \bool_if:nTF \l_@@_medium_nodes_bool
7427   {
7428     \bool_if:NTF \l_@@_no_cell_nodes_bool
7429     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7430     {
7431       \bool_if:NTF \l_@@_large_nodes_bool
7432       \@@_create_medium_and_large_nodes:
7433       \@@_create_medium_nodes:
7434     }
7435   }
7436   {
7437     \bool_if:NT \l_@@_large_nodes_bool
7438     {
7439       \bool_if:NTF \l_@@_no_cell_nodes_bool
7440       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7441       \@@_create_large_nodes:
7442     }
7443   }
7444 }
```

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`.

```

7445 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7446 {
7447   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7448   {
7449     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7450     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7451     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7452     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7453   }
7454   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7455   {
7456     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7457     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7458     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7459     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7460   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7461   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7462   {
7463     \int_step_variable:nnNn
7464     \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.

```

7465     {
7466       \cs_if_exist:cT
7467       { 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`.

```

7468     {
7469       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7470       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7471       { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7472       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7473       {
7474         \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7475         { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7476       }

```

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`.

```

7477       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7478       \dim_set:cn { l_@@_row_ \@@_i: _max_dim }
7479       { \dim_max:vn { l_@@_row_ \@@_i: _max_dim } \pgf@y }
7480       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7481       {
7482         \dim_set:cn { l_@@_column_ \@@_j: _max_dim }

```

```

7483         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7484     }
7485 }
7486 }
7487 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7488 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7489 {
7490     \dim_compare:nNnT
7491     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7492     {
7493         \@@_qpoint:n { row - \@@_i: - base }
7494         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7495         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7496     }
7497 }
7498 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7499 {
7500     \dim_compare:nNnT
7501     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7502     {
7503         \@@_qpoint:n { col - \@@_j: }
7504         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7505         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7506     }
7507 }
7508 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7509 \cs_new_protected:Npn \@@_create_medium_nodes:
7510 {
7511     \pgfpicture
7512     \pgfrememberpicturepositiononpagetrue
7513     \pgf@relevantforpicturesizefalse
7514     \@@_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”.

```

7515     \tl_set:Nn \l_@@_suffix_tl { -medium }
7516     \@@_create_nodes:
7517     \endpgfpicture
7518 }

```

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:`.

```

7519 \cs_new_protected:Npn \@@_create_large_nodes:
7520 {
7521     \pgfpicture
7522     \pgfrememberpicturepositiononpagetrue
7523     \pgf@relevantforpicturesizefalse
7524     \@@_computations_for_medium_nodes:
7525     \@@_computations_for_large_nodes:
7526     \tl_set:Nn \l_@@_suffix_tl { - large }
7527     \@@_create_nodes:

```

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7528 \endpgfpicture
7529 }
7530 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7531 {
7532 \pgfpicture
7533 \pgfrememberpicturepositiononpagetrue
7534 \pgf@relevantforpicturesizefalse
7535 \@@_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”.

```

7536 \tl_set:Nn \l_@@_suffix_tl { - medium }
7537 \@@_create_nodes:
7538 \@@_computations_for_large_nodes:
7539 \tl_set:Nn \l_@@_suffix_tl { - large }
7540 \@@_create_nodes:
7541 \endpgfpicture
7542 }

```

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.

```

7543 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7544 {
7545 \int_set:Nn \l_@@_first_row_int 1
7546 \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`.

```

7547 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7548 {
7549 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7550 {
7551 (
7552 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7553 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7554 )
7555 / 2
7556 }
7557 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7558 { l_@@_row _ \@@_i: _ min_dim }
7559 }
7560 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7561 {
7562 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7563 {
7564 (
7565 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7566 \dim_use:c
7567 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7568 )
7569 / 2
7570 }
7571 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7572 { l_@@_column _ \@@_j: _ max _ dim }
7573 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7574 \dim_sub:cn
7575 { l_@@_column _ 1 _ min _ dim }
7576 \l_@@_left_margin_dim
7577 \dim_add:cn
7578 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7579 \l_@@_right_margin_dim
7580 }

```

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).

```

7581 \cs_new_protected:Npn \@@_create_nodes:
7582 {
7583   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7584   {
7585     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7586     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7587       \@@_pgf_rect_node:nnnnn
7588       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7589       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7590       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7591       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7592       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7593       \str_if_empty:NF \l_@@_name_str
7594       {
7595         \pgfnodealias
7596         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7597         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7598       }
7599     }
7600   }
7601   \int_step_inline:nn \c@iRow
7602   {
7603     \pgfnodealias
7604     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7605     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7606   }
7607   \int_step_inline:nn \c@jCol
7608   {
7609     \pgfnodealias
7610     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7611     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7612   }
7613   \pgfnodealias
7614   { \@@_env: - last - last \l_@@_suffix_tl }
7615   { \@@_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  $n$ .

```

7616   \seq_map_pairwise_function:NNN
7617   \g_@@_multicolumn_cells_seq
7618   \g_@@_multicolumn_sizes_seq
7619   \@@_node_for_multicolumn:nn
7620 }

7621 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7622 {
7623   \cs_set_nopar:Npn \@@_i: { #1 }
7624   \cs_set_nopar:Npn \@@_j: { #2 }
7625 }

```

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”).

```

7626 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2

```

```

7627 {
7628   \@@_extract_coords_values: #1 \q_stop
7629   \@@_pgf_rect_node:nnnnn
7630   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7631   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7632   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7633   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7634   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7635   \str_if_empty:NF \l_@@_name_str
7636   {
7637     \pgfnodealias
7638     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7639     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7640   }
7641 }

```

## 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).

```

7642 \keys_define:nn { nicematrix / BlockFirstPass }
7643 {
7644   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7645   \bool_set_true:N \l_@@_p_block_bool ,
7646   j .value_forbidden:n = true ,
7647   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7648   l .value_forbidden:n = true ,
7649   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7650   r .value_forbidden:n = true ,
7651   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7652   c .value_forbidden:n = true ,
7653   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7654   L .value_forbidden:n = true ,
7655   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7656   R .value_forbidden:n = true ,
7657   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7658   C .value_forbidden:n = true ,
7659   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7660   t .value_forbidden:n = true ,
7661   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7662   T .value_forbidden:n = true ,
7663   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7664   b .value_forbidden:n = true ,
7665   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7666   B .value_forbidden:n = true ,
7667   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7668   m .value_forbidden:n = true ,
7669   v-center .meta:n = m ,
7670   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7671   p .value_forbidden:n = true ,
7672   respect-arraystretch .code:n =
7673   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7674   respect-arraystretch .value_forbidden:n = true ,

```

The following key is no longer documented in `nicematrix.tex` and `nicematrix-french.tex`. It should be considered as deprecated.

```

7675     color .code:n =
7676         \@@_color:n { #1 }
7677     \tl_set_rescan:Nnn
7678         \l_@@_draw_tl
7679         { \char_set_catcode_other:N ! }
7680         { #1 } ,
7681     color .value_required:n = true ,
7682 }

```

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.

```

7683 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7684 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7685 {

```

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).

```

7686     \tl_if_blank:nTF { #2 }
7687     { \@@_Block_ii:nnnnn 1 1 }
7688     {
7689         \tl_if_in:nnTF { #2 } { - }
7690         {
7691             \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7692             \@@_Block_i_czech:w \@@_Block_i:w
7693             #2 \q_stop
7694         }
7695         {
7696             \@@_error:nn { Bad~argument~for~Block } { #2 }
7697             \@@_Block_ii:nnnnn 1 1
7698         }
7699     }
7700     { #1 } { #3 } { #4 }
7701     \ignorespaces
7702 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7703 \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`.

```

7704 {
7705     \char_set_catcode_active:N -
7706     \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7707 }

```

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.

```

7708 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7709 {

```

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).



```

7710 \bool_lazy_or:nnTF
7711 { \tl_if_blank_p:n { #1 } }
7712 { \str_if_eq_p:ee { * } { #1 } }
7713 { \int_set:Nn \l_tmpa_int { 100 } }
7714 { \int_set:Nn \l_tmpa_int { #1 } }
7715 \bool_lazy_or:nnTF
7716 { \tl_if_blank_p:n { #2 } }
7717 { \str_if_eq_p:ee { * } { #2 } }
7718 { \int_set:Nn \l_tmpb_int { 100 } }
7719 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7720 \int_compare:nNnTF \l_tmpb_int = 1
7721 {
7722   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7723     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7724     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7725 }
7726 { \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.

```

7727 \keys_set_known:nn { nicematrix / BlockFirstPass } { #3 }
7728 \tl_set:Ne \l_tmpa_tl
7729 {
7730   { \int_use:N \c@iRow }
7731   { \int_use:N \c@jCol }
7732   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7733   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7734 }

```

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).

```

7735 \bool_set_false:N \l_tmpa_bool
7736 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7737 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7738 \bool_case:nF
7739 {
7740   \l_tmpa_bool { \@@_Block_vii:eennn }
7741   \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7742 \l_@@_X_bool { \@@_Block_v:eennn }
7743 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7744 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7745 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7746 }
7747 { \@@_Block_v:eennn }
7748 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7749 }

```

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.

```

7750 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7751 {
7752   \int_gincr:N \g_@@_block_box_int
7753   \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7754   \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7755   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7756   {
7757     \tl_gput_right:Ne \g_@@_rules_tl
7758     {
7759       \@@_draw_diagbox:nnnnnn
7760       { \int_use:N \c@iRow }
7761       { \int_use:N \c@jCol }
7762       { \int_eval:n { \c@iRow + #1 - 1 } }
7763       { \int_eval:n { \c@jCol + #2 - 1 } }
7764       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7765       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7766     }
7767   }
7768   \box_gclear_new:c
7769   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7770   \hbox_gset:cn
7771   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7772   {

```

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`).

```

7773   \tl_if_empty:NTF \l_@@_color_tl
7774   { \int_compare:nNnT { #2 } = 1 \set@color }
7775   { \@@_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`.

```

7776   \int_compare:nNnT { #1 } = 1
7777   {
7778     \int_if_zero:nTF \c@iRow
7779     {

```

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}$

```

```

7780 \cs_set_eq:NN \Block \@@_NullBlock:
7781 \l_@@_code_for_first_row_tl
7782 }
7783 {
7784 \int_compare:nNt \c@iRow = \l_@@_last_row_int
7785 {
7786 \cs_set_eq:NN \Block \@@_NullBlock:
7787 \l_@@_code_for_last_row_tl
7788 }
7789 }
7790 \g_@@_row_style_tl
7791 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7792 \@@_reset_arraystretch:
7793 \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7794 #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.).

```

7795 \@@_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}`.

```

7796 \bool_if:NTF \l_@@_tabular_bool
7797 {
7798 \bool_lazy_all:nTF
7799 {
7800 { \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.

```

7801 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7802 { ! \g_@@_rotate_bool }
7803 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7804 {
7805 \use:e
7806 {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7807 \exp_not:N \begin { minipage }
7808 [ \str_lowercase:f \l_@@_vpos_block_str ]
7809 { \l_@@_col_width_dim }
7810 \str_case:on \l_@@_hpos_block_str
7811 { c \centering r \raggedleft l \raggedright }

```

```

7812         }
7813         #5
7814     \end { minipage }
7815 }

```

In the other cases, we use a `{tabular}`.

```

7816     {
7817         \use:e
7818         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7819         \exp_not:N \begin { tabular }
7820         [ \str_lowercase:f \l_@@_vpos_block_str ]
7821         { @ { } \l_@@_hpos_block_str @ { } }
7822     }
7823     #5
7824     \end { tabular }
7825 }
7826 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7827     {
7828         $ % $
7829         \bool_if:NT \l_@@_small_bool % 2026/04/05
7830         {
7831             \def \arraystretch { 0.47 }
7832             \dim_set:Nn \arraycolsep { 1.45 pt }
7833         }
7834         \use:e
7835         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7836         \exp_not:N \begin { array }
7837         [ \str_lowercase:f \l_@@_vpos_block_str ]
7838         { @ { } \l_@@_hpos_block_str @ { } }
7839     }
7840     \@@_tuning_key_small: % 2026/04/05
7841     #5
7842     \end { array }
7843     $ % $
7844 }
7845 }

```

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).

```

7846     \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.

```

7847     \int_compare:nNtT { #2 } = 1
7848     {
7849         \dim_gset:Nn \g_@@_blocks_wd_dim
7850         {
7851             \dim_max:nn
7852             \g_@@_blocks_wd_dim
7853             {
7854                 \box_wd:c
7855                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7856             }
7857         }
7858     }

```

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.

```

7859   \int_compare:nNnT { #1 } = 1
7860   {
7861     \bool_lazy_any:nT
7862     {
7863       { \str_if_empty_p:N \l_@@_vpos_block_str }
7864       { \str_if_eq_p:ee t \l_@@_vpos_block_str }
7865       { \str_if_eq_p:ee b \l_@@_vpos_block_str }
7866     }
7867     \@@_adjust_blocks_ht_dp:
7868   }
7869   \seq_gput_right:Ne \g_@@_blocks_seq
7870   {
7871     \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.

```

7872   {
7873     \exp_not:n { #3 } ,
7874     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7875     \bool_if:NT \g_@@_rotate_bool
7876     {
7877       \bool_if:NTF \g_@@_rotate_c_bool
7878       { m }
7879       {
7880         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7881         { T }
7882       }
7883     }
7884   }
7885   {
7886     \box_use_drop:c
7887     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7888   }
7889 }
7890 \bool_set_false:N \g_@@_rotate_c_bool
7891 }

7892 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7893 {
7894   \dim_gset:Nn \g_@@_blocks_ht_dim
7895   {
7896     \dim_max:nn
7897     \g_@@_blocks_ht_dim
7898     {
7899       \box_ht:c
7900       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7901     }
7902   }
7903   \dim_gset:Nn \g_@@_blocks_dp_dim
7904   {
7905     \dim_max:nn
7906     \g_@@_blocks_dp_dim
7907     {
7908       \box_dp:c
7909       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7910     }

```

```

7911     }
7912 }

7913 \cs_new:Npn \@@_adjust_hpos_rotate:
7914 {
7915     \bool_if:NT \g_@@_rotate_bool
7916     {
7917         \str_set:Ne \l_@@_hpos_block_str
7918         {
7919             \bool_if:NTF \g_@@_rotate_c_bool
7920             c
7921             {
7922                 \str_case:onF \l_@@_vpos_block_str
7923                 { b l B l t r T r }
7924                 {
7925                     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7926                     r
7927                     l
7928                 }
7929             }
7930         }
7931     }
7932 }
7933 \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*.

```

7934 \cs_new_protected:Npn \@@_rotate_box_of_block:
7935 {
7936     \box_grotate:cn
7937     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7938     { \bool_if:NTF \g_@@_rotate_minus_bool { -90 } { 90 } }
7939     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
7940     {
7941         \vbox_gset_top:cn
7942         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7943         {
7944             \skip_vertical:n { 0.8 ex }
7945             \box_use:c
7946             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7947         }
7948     }
7949     \bool_if:NT \g_@@_rotate_c_bool
7950     {
7951         \hbox_gset:cn
7952         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7953         {
7954             \IfFormatAtLeastTF { 2026-04-01 }
7955             {
7956                 \vbox_center:n
7957                 {
7958                     \box_use:c
7959                     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7960                 }
7961             }
7962             {
7963                 $ % $
7964                 \vcenter
7965                 {
7966                     \box_use:c
7967                     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7968                 }

```

```

7969             $ % $
7970         }
7971     }
7972 }
7973 }

```

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.

```

7974 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7975 {
7976   \seq_gput_right:Ne \g_@@_blocks_seq
7977   {
7978     \l_tmpa_tl
7979     { \exp_not:n { #3 } }
7980     {
7981       \bool_if:NTF \l_@@_tabular_bool
7982       {
7983         {

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7984 \@@_reset_arraystretch:
7985 \exp_not:n
7986 {
7987   \dim_zero:N \extrarowheight
7988   #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).

```

7989 \tag_if_active:T { \tag_stop:n { table } }
7990 \use:e
7991 {
7992   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7993   { @ { } \l_@@_hpos_block_str @ { } }
7994 }
7995 #5
7996 \end { tabular }
7997 }
7998 }
7999 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

8000 {
8001 }

```

The following will be no-op when `respect-arraystretch` is in force.

```

8002 \@@_reset_arraystretch:
8003 \exp_not:n
8004 {
8005   \dim_zero:N \extrarowheight
8006   #4
8007   $ % $
8008   \use:e
8009   {
8010     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
8011     { @ { } \l_@@_hpos_block_str @ { } }
8012   }

```

```

8013         \@@_tuning_key_small: % 2026/05/04
8014         #5
8015     \end { array }
8016     $ % $
8017 }
8018 }
8019 }
8020 }
8021 }
8022 }
8023 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

8024 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
8025 {
8026     \seq_gput_right:Ne \g_@@_blocks_seq
8027     {
8028         \l_tmpa_tl
8029         { \exp_not:n { #3 } }
8030         { { \exp_not:n { #4 #5 } } }
8031     }
8032 }
8033 \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`.

```

8034 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
8035 {
8036     \seq_gput_right:Ne \g_@@_blocks_seq
8037     {
8038         \l_tmpa_tl
8039         { \exp_not:n { #3 } }
8040         { \exp_not:n { #4 #5 } }
8041     }
8042 }
8043 \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).

```

8044 \keys_define:nn { nicematrix / Block }
8045 {
8046     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
8047     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

8048     tikz .code:n =
8049     \IfPackageLoadedTF { tikz }
8050     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
8051     { \@@_error:n { tikz-key-without-tikz } } ,
8052     tikz .value_required:n = true ,
8053     fill .code:n =
8054     \tl_set_rescan:Nnn
8055     \l_@@_fill_tl
8056     { \char_set_catcode_other:N ! }
8057     { #1 } ,
8058     fill .value_required:n = true ,

```



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

```

8059 opacity .tl_set:N = \l_@@_opacity_tl ,
8060 opacity .value_required:n = true ,
8061 draw .code:n =
8062   \tl_set_rescan:Nnn
8063   \l_@@_draw_tl
8064   { \char_set_catcode_other:N ! }
8065   { #1 } ,
8066 draw .default:n = default ,
8067 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8068 rounded-corners .default:n = 4 pt ,
8069 color .code:n =
8070   \@@_color:n { #1 }
8071   \tl_set_rescan:Nnn
8072   \l_@@_draw_tl
8073   { \char_set_catcode_other:N ! }
8074   { #1 } ,
8075 borders .clist_set:N = \l_@@_borders_clist ,
8076 borders .default:n = all , % added 2026/05/08
8077 hvlines .meta:n = { vlines , hlines } ,
8078 vlines .bool_set:N = \l_@@_vlines_block_bool ,
8079 hlines .bool_set:N = \l_@@_hlines_block_bool ,
8080 rules/width .dim_set:N = \arrayrulewidth ,
8081 rules/color .tl_set:N = \l_@@_rules_color_tl ,
8082 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,

```

The key `line-width` is now deprecated (replaced by `rules/width`).

```

8083 line-width .dim_set:N = \arrayrulewidth , % deprecated

```

Some keys have not a property `.value_required:n` (or similar) because they are in `BlockFirstPass`.

```

8084 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
8085   \bool_set_true:N \l_@@_p_block_bool ,
8086 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
8087 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
8088 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
8089 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
8090   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8091 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
8092   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8093 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
8094   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
8095 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
8096 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
8097 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
8098 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
8099 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
8100 m .value_forbidden:n = true ,
8101 v-center .meta:n = m ,
8102 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
8103 p .value_forbidden:n = true ,
8104 name .str_set:N = \l_@@_block_name_str ,
8105 name .value_required:n = true ,
8106 respect-arraystretch .code:n =
8107   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
8108 respect-arraystretch .value_forbidden:n = true ,
8109 transparent .bool_set:N = \l_@@_transparent_bool ,
8110 unknown .code:n =
8111   \@@_unknown_key:nn
8112   { nicematrix / Block }
8113   { Unknown~key~for~Block }
8114 }

```

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.

```

8115 \cs_new_protected:Npn \@@_draw_blocks:
8116 {
8117   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
8118   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
8119 }

8120 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
8121 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

8122   \int_zero:N \l_@@_last_row_int
8123   \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”).

```

8124   \int_compare:nNnTF { #3 } > { 98 }
8125   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
8126   { \int_set:Nn \l_@@_last_row_int { #3 } }
8127   \int_compare:nNnTF { #4 } > { 98 }
8128   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
8129   { \int_set:Nn \l_@@_last_col_int { #4 } }

8130   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
8131   {
8132     \bool_lazy_and:nnTF
8133       \l_@@_preamble_bool
8134       {
8135         \int_compare_p:n
8136           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
8137       }
8138     {
8139       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
8140       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
8141       \@@_msg_redirect_name:nn { columns-not-used } { none }
8142     }
8143     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8144   }
8145   {
8146     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
8147     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
8148     {

```

We expand the four first arguments of `\@@_Block_v:nnnnnn`.

```

8149       \use:e
8150       {
8151         \@@_Block_v:nnnnnn
8152         { #1 }
8153         { #2 }
8154         { \int_use:N \l_@@_last_row_int }
8155         { \int_use:N \l_@@_last_col_int }
8156       }
8157       { #5 }
8158       { #6 }
8159     }
8160   }
8161 }

```

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.

```
8162 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
8163 {
```

The group is for the keys.

```
8164 \group_begin:
8165 \int_compare:nNt { #1 } = { #3 }
8166 { \str_set:Nn \l_@@_vpos_block_str { t } }
8167 \keys_set:nn { nicematrix / Block } { #5 }
```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```
8168 \bool_if:NT \l_@@_amp_in_blocks_bool
8169 { \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool } }

8170 \bool_lazy_and:nnT
8171 \l_@@_vlines_block_bool
8172 { ! \l_@@_ampersand_bool }
8173 {
8174   \tl_gput_right:Ne \g_@@_rules_tl
8175   {
8176     \@@_vlines_block:nnnnnn
8177     { \dim_use:N \arrayrulewidth }
8178     { \l_@@_rules_color_tl }
8179     { #1 } { #2 } { #3 } { #4 }
8180   }
8181 }

8182 \bool_if:NT \l_@@_hlines_block_bool
8183 {
8184   \tl_gput_right:Ne \g_@@_rules_tl
8185   {
8186     \@@_hlines_block:nnnnnn
8187     { \dim_use:N \arrayrulewidth }
8188     { \l_@@_rules_color_tl }
8189     { #1 } { #2 } { #3 } { #4 }
8190   }
8191 }
```

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).

```
8192 \bool_if:NF \l_@@_transparent_bool
8193 {
8194   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8195   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
8196 }

8197 \tl_if_empty:NF \l_@@_draw_tl
8198 {
8199   \tl_gput_right:Nn \g_@@_rules_tl
8200   {
8201     \@@_stroke_block:nnnnn
```

#5 are the options

```
8202     { #5 } { #1 } { #2 } { #3 } { #4 }
8203   }
8204   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
8205   { { #1 } { #2 } { #3 } { #4 } }
8206 }

8207 \clist_if_empty:NF \l_@@_borders_clist
8208 {
8209   \tl_gput_right:Nn \g_nicematrix_code_after_tl
8210   {
```

```

8211         \@@_stroke_borders_block:nnnnn
8212         { #5 } { #1 } { #2 } { #3 } { #4 }
8213     }
8214 }
8215 \tl_if_empty:NF \l_@@_fill_tl
8216 {
8217     \@@_add_opacity_to_fill:
8218     \tl_gput_right:Ne \g_@@_pre_code_before_tl
8219     {
8220         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
8221         { #1 - #2 }
8222         { #3 - #4 }
8223         { \dim_use:N \l_@@_rounded_corners_dim }
8224     }
8225 }
8226 \seq_if_empty:NF \l_@@_tikz_seq
8227 {
8228     \tl_gput_right:Ne \g_nicematrix_code_before_tl
8229     {
8230         \@@_block_tikz:nnnnn
8231         { \seq_use:Nn \l_@@_tikz_seq { , } }
8232         { #1 } { #2 } { #3 } { #4 }

```

We will have in that last field a list of lists of TikZ keys.

```

8233     }
8234 }
8235 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
8236 {
8237     \tl_gput_right:Ne \g_@@_rules_tl
8238     {
8239         \@@_draw_diagbox:nnnnnn
8240         { #1 } { #2 } { #3 } { #4 }
8241         { \exp_not:n { ##1 } }
8242         { \exp_not:n { ##2 } }
8243     }
8244 }

```

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.

```

8245 \pgfpicture
8246 \pgfrememberpicturepositiononpagetrue
8247 \pgf@relevantforpicturesizefalse

```

```

8248 \@@_qpoint:n { row - #1 }
8249 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8250 \@@_qpoint:n { col - #2 }
8251 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8252 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
8253 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8254 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8255 \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.

```

8256 \@@_pgf_rect_node:nnnnn
8257 { \@@_env: - #1 - #2 - block }
8258 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8259 \str_if_empty:NF \l_@@_block_name_str
8260 {
8261   \pgfnodealias
8262   { \@@_env: - \l_@@_block_name_str }
8263   { \@@_env: - #1 - #2 - block }
8264   \str_if_empty:NF \l_@@_name_str
8265   {
8266     \pgfnodealias
8267     { \l_@@_name_str - \l_@@_block_name_str }
8268     { \@@_env: - #1 - #2 - block }
8269   }
8270 }

```

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.

```

8271 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8272 {
8273   \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.

```

8274 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8275 {

```

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.

```

8276 \cs_if_exist:cT
8277 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8278 {
8279   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8280   {
8281     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8282     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8283   }
8284 }
8285 }

```

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.

```

8286 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
8287 {
8288   \@@_qpoint:n { col - #2 }
8289   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8290 }
8291 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8292 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8293 {

```

```

8294 \cs_if_exist:cT
8295 { pgf @ sh @ ns @ \@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8296 {
8297   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8298   {
8299     \pgfpointanchor
8300     { \@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8301     { east }
8302     \dim_set:Nn \l_@@_tmpd_dim
8303     { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
8304   }
8305 }
8306 }
8307 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
8308 {
8309   \@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8310   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8311 }
8312 \@_pgf_rect_node:nnnnn
8313 { \@_env: - #1 - #2 - block - short }
8314 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8315 }

```

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.

```

8316 \bool_if:NT \l_@@_medium_nodes_bool
8317 {
8318   \@_pgf_rect_node:nnn
8319   { \@_env: - #1 - #2 - block - medium }
8320   { \pgfpointanchor { \@_env: - #1 - #2 - medium } { north-west } }
8321   {
8322     \pgfpointanchor
8323     { \@_env:
8324       - \int_use:N \l_@@_last_row_int
8325       - \int_use:N \l_@@_last_col_int - medium
8326     }
8327     { south-east }
8328   }
8329 }
8330 \endpgfpicture
8331

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8332 \bool_if:NTF \l_@@_ampersand_bool
8333 {
8334   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8335   \int_zero_new:N \l_@@_split_int
8336   \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).

```

8337 \int_set:Nn \l_@@_first_row_int { #1 }
8338 \int_set:Nn \l_@@_first_col_int { #2 }
8339 \int_set:Nn \l_@@_last_row_int { #3 }
8340 \int_set:Nn \l_@@_last_col_int { #4 }
8341
8342 \pgfpicture
8343 \pgfrememberpicturepositiononpagetrue
8344 \pgf@relevantforpicturesizefalse
8345 \@_qpoint:n { row - #1 }
8346 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8347 \@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8348 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y

```

```

8348 \@@_qpoint:n { col - #2 }
8349 \dim_set_eq:NN \l_tmpa_dim \pgf@x
8350 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8351 \dim_set:Nn \l_tmpb_dim
8352 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8353 \bool_lazy_or:nnT
8354 \l_@@_vlines_block_bool
8355 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
8356 {
8357   \int_step_inline:nn { \l_@@_split_int - 1 }
8358   {
8359     \pgfpathmoveto
8360     {
8361       \pgfpoint
8362       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8363       \l_@@_tmpc_dim
8364     }
8365     \pgfpathlineto
8366     {
8367       \pgfpoint
8368       { \l_tmpa_dim + ##1 \l_tmpb_dim }
8369       \l_@@_tmpd_dim
8370     }
8371     \CT@arc@
8372     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8373     \pgfsetrectcap
8374     \pgfusepathqstroke
8375   }
8376 }
8377 \cs_set_eq:NN \cellcolor \@@_subcellcolor
8378 \int_zero_new:N \l_@@_split_i_int
8379 \str_if_eq:eeTF \l_@@_vpos_block_str T
8380 {
8381   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8382   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8383 }
8384 {
8385   \str_if_eq:eeTF \l_@@_vpos_block_str B
8386   {
8387     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8388     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8389   }
8390   {
8391     \bool_lazy_or:nnTF
8392     { \int_compare_p:nNn { #1 } = { #3 } }
8393     { \str_if_eq_p:ee \l_@@_vpos_block_str t }
8394     {
8395       \@@_qpoint:n { row - #1 - base }
8396       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8397     }
8398     {
8399       \str_if_eq:eeTF \l_@@_vpos_block_str b
8400       {
8401         \@@_qpoint:n { row - #3 - base }
8402         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8403       }
8404       {
8405         \@@_qpoint:n { #1 - #2 - block }
8406         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8407       }
8408     }
8409   }
8410 }

```

```

8411 \int_step_inline:nn \l_@@_split_int
8412 {
8413   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8414   \int_set:Nn \l_@@_split_i_int { ##1 }
8415   \dim_set:Nn \col@sep
8416     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
8417   \pgftransformshift
8418     {
8419     \pgfpoint
8420     {
8421       \l_tmpa_dim + ##1 \l_tmpb_dim -
8422       \str_case:on \l_@@_hpos_block_str
8423         {
8424           l { \l_tmpb_dim + \col@sep }
8425           c { 0.5 \l_tmpb_dim }
8426           r { \col@sep }
8427         }
8428       }
8429     { \l_@@_tmpc_dim }
8430   }
8431   \pgfset { inner~sep = \c_zero_dim }
8432   \pgfnode
8433     { rectangle }
8434     {
8435       \str_if_eq:eeTF T \l_@@_vpos_block_str
8436       {
8437         \str_case:on \l_@@_hpos_block_str
8438           {
8439             l { north-west }
8440             c { north }
8441             r { north-east }
8442           }
8443       }
8444     {
8445       \str_if_eq:eeTF B \l_@@_vpos_block_str
8446       {
8447         \str_case:on \l_@@_hpos_block_str
8448           {
8449             l { south-west }
8450             c { south }
8451             r { south-east }
8452           }
8453       }
8454     {
8455       \bool_lazy_any:nTF
8456       {
8457         { \int_compare_p:nNn { #1 } = { #3 } }
8458         { \str_if_eq_p:ee t \l_@@_vpos_block_str }
8459         { \str_if_eq_p:ee b \l_@@_vpos_block_str }
8460       }
8461       {
8462         \str_case:on \l_@@_hpos_block_str
8463           {
8464             l { base-west }
8465             c { base }
8466             r { base-east }
8467           }
8468       }
8469     {
8470       \str_case:on \l_@@_hpos_block_str
8471       {
8472         l { west }

```



```

8473             c { center }
8474             r { east }
8475         }
8476     }
8477 }
8478 }
8479 }
8480 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8481 \group_end:
8482 }
8483 \endpgfpicture
8484 }

```

Now the case where there is no ampersand & in the content of the block.

```

8485 {
8486   \bool_if:NTF \l_@@_p_block_bool
8487   {

```

When the final user has used the key p, we have to compute the width.

```

8488     \pgfpicture
8489     \pgfrememberpicturepositiononpagetrue
8490     \pgf@relevantforpicturesizefalse
8491     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8492     {
8493       \@@_qpoint:n { col - #2 }
8494       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8495       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8496     }
8497     {
8498       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8499       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8500       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8501     }
8502     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8503   \endpgfpicture
8504   \hbox_set:Nn \l_@@_cell_box
8505   {
8506     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8507       { \g_tmpb_dim }
8508     \str_case:on \l_@@_hpos_block_str
8509       { c \centering r \raggedleft l \raggedright j { } }
8510     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8511     #6
8512     \end { minipage }
8513   }
8514 }
8515 {
8516   \hbox_set:Nn \l_@@_cell_box
8517   {
8518     \set@color
8519     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
8520     #6
8521   }
8522 }
8523 \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.

```

8524   \pgfpicture
8525   \pgfrememberpicturepositiononpagetrue
8526   \pgf@relevantforpicturesizefalse
8527   \bool_lazy_any:nTF
8528   {
8529     { \str_if_empty_p:N \l_@@_vpos_block_str }

```

```

8530         { \str_if_eq_p:ee c \l_@@_vpos_block_str }
8531         { \str_if_eq_p:ee T \l_@@_vpos_block_str }
8532         { \str_if_eq_p:ee B \l_@@_vpos_block_str }
8533     }

```

```

8534 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8535     \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`.

```

8536     \bool_if:nT \g_@@_last_col_found_bool
8537     {
8538         \int_compare:nNnT { #2 } = \g_@@_col_total_int
8539         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8540     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8541     \tl_set:Ne \l_tmpa_tl
8542     {
8543         \str_case:on \l_@@_vpos_block_str
8544         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8545         { } {
8546             \str_case:on \l_@@_hpos_block_str
8547             {
8548                 c { center }
8549                 l { west }
8550                 r { east }
8551                 j { center }
8552             }
8553         }
8554     c {
8555         \str_case:on \l_@@_hpos_block_str
8556         {
8557             c { center }
8558             l { west }
8559             r { east }
8560             j { center }
8561         }
8562     }
8563 }
8564 T {
8565     \str_case:on \l_@@_hpos_block_str
8566     {
8567         c { north }
8568         l { north~west }
8569         r { north~east }
8570         j { north }
8571     }
8572 }
8573 }
8574 B {
8575     \str_case:on \l_@@_hpos_block_str
8576     {
8577         c { south }
8578         l { south~west }
8579         r { south~east }
8580         j { south }
8581     }
8582 }
8583 }
8584 }

```

```

8585     }
8586     \pgftransformshift
8587     {
8588         \pgfpointanchor
8589         {
8590             \@@_env: - #1 - #2 - block
8591             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8592         }
8593         \l_tmpa_tl
8594     }
8595     \pgfset { inner~sep = \c_zero_dim }
8596     \pgfnode
8597     { rectangle }
8598     \l_tmpa_tl
8599     { \box_use_drop:N \l_@@_cell_box } { } { }
8600 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8601 {
8602     \pgfextracty \l_tmpa_dim
8603     {
8604         \@@_qpoint:n
8605         {
8606             row - \str_if_eq:eeTF b \l_@@_vpos_block_str { #3 } { #1 }
8607             - base
8608         }
8609     }
8610     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

8611     \pgfpointanchor
8612     {
8613         \@@_env: - #1 - #2 - block
8614         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8615     }
8616     {
8617         \str_case:on \l_@@_hpos_block_str
8618         {
8619             c { center }
8620             l { west }
8621             r { east }
8622             j { center }
8623         }
8624     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8625     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8626     \pgfset { inner~sep = \c_zero_dim }
8627     \pgfnode
8628     { rectangle }
8629     {
8630         \str_case:on \l_@@_hpos_block_str
8631         {
8632             c { base }
8633             l { base~west }
8634             r { base~east }
8635             j { base }
8636         }
8637     }
8638     { \box_use_drop:N \l_@@_cell_box } { } { }
8639 }
8640 \endpgfpicture
8641 }

```

```

8642 \group_end:
8643 }
8644 \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).

```

8645 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8646 {
8647   \tl_if_empty:NF \l_@@_opacity_tl
8648   {
8649     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8650       {
8651         \tl_set:Nc \l_@@_fill_tl
8652           {
8653             [ opacity = \l_@@_opacity_tl ,
8654             \tl_tail:o \l_@@_fill_tl
8655           }
8656       }
8657       {
8658         \tl_set:Nc \l_@@_fill_tl
8659           { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8660       }
8661     }
8662   }

```

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*.

```

8663 \cs_new_protected:Npn \@@_stroke_block:nnnnn #1 #2 #3 #4 #5
8664 {
8665   \group_begin:
8666   \tl_clear:N \l_@@_draw_tl
8667   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8668   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8669   \tl_if_empty:NF \l_@@_draw_tl
8670   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8671     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8672       \CT@arc@
8673       { \@@_color:o \l_@@_draw_tl }
8674   }

```

The following code can't be put just before the `\pgfusepath { stroke }` (it's too late).

```

8675 \pgfsetcornersarced
8676 { \pgfpoint \l_@@_rounded_corners_dim \l_@@_rounded_corners_dim }
8677 \int_compare:nNnF { #2 } > \c@iRow
8678 {
8679   \int_compare:nNnF { #3 } > \c@jCol
8680   {
8681     \@@_qpoint:n { row - #2 }
8682     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8683     \@@_qpoint:n { col - #3 }
8684     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8685     \@@_qpoint:n
8686       { row - \int_eval:n { 1 + \int_min:nn \c@iRow { #4 } } }
8687     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8688     \@@_qpoint:n
8689       { col - \int_eval:n { 1 + \int_min:nn \c@jCol { #5 } } }
8690     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8691     \pgfpathrectanglecorners
8692       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }

```

```

8693         { \pgfpoint \pgf@x \l_tmpa_dim }
8694         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8695             \pgfusepathqstroke
8696         { \pgfusepath { stroke } }
8697     }
8698 }
8699 \group_end:
8700 }

```

Here is the set of keys for the command `\@@_stroke_block:nnnnn`.

```

8701 \keys_define:nn { nicematrix / BlockStroke }
8702 {
8703     color .tl_set:N = \l_@@_draw_tl ,
8704     draw .code:n =
8705         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8706     draw .default:n = default ,
8707     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8708     rounded-corners .default:n = 4 pt ,
8709     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The key `line-width` is now deprecated.

```

8710     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8711 }

```

The command `\@@_vlines_block:nnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draw the block.

The first argument of `\@@_vlines_block:nnn` is the width of the rules that we will draw. The second is the color. The other arguments are the position of the block: *imin*, *jmin*, *imax* and *jmax*.

```

8712 \cs_new_protected:Npn \@@_vlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8713 {
8714     \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).

```

8715     \dim_set:Nn \arrayrulewidth { #1 }
8716     \pgfsetlinewidth \arrayrulewidth % added 2026-03-28
8717     \@@_set_CTarc:n { #2 }
8718     \CT@arc@

```

We filter the list of blocks `\g_@@_pos_of_blocks_seq` in order to discard the blocks which encompass the current block (elsewhere, of course, the rules would not be drawn).

```

8719     \seq_set_filter:NnN \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8720     {
8721         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8722         ||
8723         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8724         ||
8725         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8726         ||
8727         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8728     }
8729     \bool_if:NT \l_@@_fix_vertex_bool
8730     {
8731         \int_compare:nNnT { #4 } > 1
8732         {
8733             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8734             {
8735                 { 1 }
8736                 { 1 }
8737                 { \int_use:N \c@iRow }
8738                 { \int_eval:n { #4 -1 } }

```

```

8739         { }
8740     }
8741 }
8742 \int_compare:nNt { #6 } < \c@jCol
8743 {
8744     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8745     {
8746         { 1 }
8747         { \int_eval:n { #6 + 1 } }
8748         { \int_use:N \c@iRow }
8749         { \int_use:N \c@jCol }
8750     }
8751 }
8752 }
8753 }
8754 \int_set:Nn \l_@@_start_int { #3 }
8755 \int_set:Nn \l_@@_end_int { #5 }
8756 \int_step_inline:nnn { #4 } { #6 + 1 }
8757 {
8758     \int_set:Nn \l_@@_position_int { ##1 }
8759     \socket_use:n { nicematrix / draw-vrule }
8760 }
8761 \group_end:
8762 }

```

The command `\@@_hlines_block:nnnnnn` is used only once: in `\@@_Block_v:nnnnnn` which actually draws the block.

```

8763 \cs_new_protected:Npn \@@_hlines_block:nnnnnn #1 #2 #3 #4 #5 #6
8764 {
8765     \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).

```

8766     \dim_set:Nn \arrayrulewidth { #1 }
8767     \pgfsetlinewidth \arrayrulewidth % added 2026/03/28
8768     \@@_set_CTarc:n { #2 }
8769     \CT@arc@

```

We filter the list of blocks `\g_@@_pos_of_blocks_seq` in order to discard the blocks which encompass the current block (elsewhere, of course, the rules would not be drawn).

```

8770     \seq_set_filter:Nn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq % noqa: E420
8771     {
8772         \int_compare_p:nNn { \use_i:nnnnn ##1 } > { #3 }
8773         ||
8774         \int_compare_p:nNn { \use_ii:nnnnn ##1 } > { #4 }
8775         ||
8776         \int_compare_p:nNn { \use_iii:nnnnn ##1 } < { #5 }
8777         ||
8778         \int_compare_p:nNn { \use_iv:nnnnn ##1 } < { #6 }
8779     }
8780     \bool_if:NT \l_@@_fix_vertex_bool
8781     {
8782         \int_compare:nNt { #3 } > 1
8783         {
8784             \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8785             {
8786                 { 1 }
8787                 { 1 }
8788                 { \int_eval:n { #3 - 1 } }
8789                 { \int_use:N \c@jCol }

```

```

8790         { }
8791     }
8792 }
8793 \int_compare:nNt { #5 } < \c@iRow
8794 {
8795     \seq_put_right:Ne \g_@@_pos_of_blocks_seq
8796     {
8797         { \int_eval:n { #5 + 1 } }
8798         { 1 }
8799         { \int_use:N \c@iRow }
8800         { \int_use:N \c@jCol }
8801         { }
8802     }
8803 }
8804 }
8805 \int_set:Nn \l_@@_start_int { #4 }
8806 \int_set:Nn \l_@@_end_int { #6 }
8807 \int_step_inline:nnn { #3 } { #5 + 1 }
8808 {
8809     \int_set:Nn \l_@@_position_int { ##1 }
8810     \socket_use:n { nicematrix / draw-hrule }
8811 }
8812 \group_end:
8813 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke.

```

8814 \cs_new_protected:Npn \@@_stroke_borders_block:nnnnn #1 #2 #3 #4 #5
8815 {
8816     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8817     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8818     \dim_compare:nNtF \l_@@_rounded_corners_dim > \c_zero_dim
8819     { \@@_error:n { borders~forbidden } }
8820     {
8821         \tl_clear_new:N \l_@@_borders_tikz_tl
8822         \keys_set:no { nicematrix / OnlyForTikzInBorders } \l_@@_borders_clist
8823         \tl_set:Nn \l_@@_tmpc_tl { #2 }
8824         \tl_set:Nn \l_@@_tmpd_tl { #3 }
8825         \tl_set:Ne \l_tmpa_tl { \int_eval:n { #4 + 1 } }
8826         \tl_set:Ne \l_tmpb_tl { \int_eval:n { #5 + 1 } }
8827         \@@_stroke_borders_block_i:
8828     }
8829 }
8830 \AtBeginDocument
8831 {
8832     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8833     {
8834         \c_@@_pgfortikzpicture_tl
8835         \@@_stroke_borders_block_ii:
8836         \c_@@_endpgfortikzpicture_tl
8837     }
8838 }
8839 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8840 {
8841     \pgfrememberpicturepositiononpagetrue
8842     \pgf@relevantforpicturesizefalse
8843     \CT@arc@
8844     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8845     \clist_if_in:NnTF \l_@@_borders_clist { all }
8846     {
8847         \@@_stroke_vertical:n \l_tmpb_tl
8848         \@@_stroke_vertical:n \l_@@_tmpd_tl

```

```

8849     \@@_stroke_horizontal:n \l_tmpa_tl
8850     \@@_stroke_horizontal:n \l_@@_tmpc_tl
8851   }
8852   {
8853     \clist_if_in:NnT \l_@@_borders_clist { right }
8854     { \@@_stroke_vertical:n \l_tmpb_tl }
8855     \clist_if_in:NnT \l_@@_borders_clist { left }
8856     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8857     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8858     { \@@_stroke_horizontal:n \l_tmpa_tl }
8859     \clist_if_in:NnT \l_@@_borders_clist { top }
8860     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8861   }
8862 }
8863 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8864 {
8865   tikz .code:n =
8866     \cs_if_exist:NTF \tikzpicture
8867     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8868     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8869   tikz .value_required:n = true ,
8870   top .code:n = ,
8871   bottom .code:n = ,
8872   left .code:n = ,
8873   right .code:n = ,
8874   all .code:n = ,
8875   unknown .code:n = \@@_error:n { bad~border }
8876 }

```

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).

```

8877 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8878 {
8879   \@@_qpoint:n \l_@@_tmpc_tl
8880   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8881   \@@_qpoint:n \l_tmpa_tl
8882   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8883   \@@_qpoint:n { #1 }
8884   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8885   {
8886     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8887     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8888     \pgfusepathqstroke
8889   }
8890   {
8891     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8892     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8893   }
8894 }

```

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).

```

8895 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8896 {
8897   \@@_qpoint:n \l_@@_tmpd_tl
8898   \clist_if_in:NnTF \l_@@_borders_clist { left }
8899   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8900   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8901   \@@_qpoint:n \l_tmpb_tl
8902   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8903   \@@_qpoint:n { #1 }
8904   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8905   {

```



```

8906     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8907     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8908     \pgfusepathqstroke
8909 }
8910 {
8911     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8912     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8913 }
8914 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8915 \keys_define:nn { nicematrix / BlockBorders }
8916 {
8917     borders .clist_set:N = \l_@@_borders_clist ,
8918     borders .default:n = all , % 2026/08/05
8919     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8920     rounded-corners .default:n = 4 pt ,
8921     rules/width .dim_set:N = \l_@@_line_width_dim ,

```

The following key is deprecated.

```

8922     line-width .dim_set:N = \l_@@_line_width_dim % deprecated
8923 }

```

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.

```

8924 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8925 {
8926     \begin { tikzpicture }
8927     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8928     \clist_map_inline:nn { #1 }
8929     {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8930     \keys_set_known:nnN { nicematrix / SpecialOffset } { #1 } \l_tmpa_tl
8931     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8932     (
8933     [
8934         xshift = \dim_use:N \l_@@_offset_dim ,
8935         yshift = - \dim_use:N \l_@@_offset_dim
8936     ]
8937     #2 -| #3
8938     )
8939     rectangle
8940     (
8941     [
8942         xshift = - \dim_use:N \l_@@_offset_dim ,
8943         yshift = \dim_use:N \l_@@_offset_dim
8944     ]
8945     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8946     ) ;
8947 }
8948 \end { tikzpicture }
8949 }
8950 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8951 \keys_define:nn { nicematrix / SpecialOffset }
8952 {
8953   offset .dim_set:N = \l_@@_offset_dim ,
8954 }

```

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.

```

8955 \cs_new_protected:Npn \@@_NullBlock:
8956 { \@@_collect_options:n { \@@_NullBlock_i: } }
8957 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8958 { }

```

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.

```

8959 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8960 {
8961   \tl_gput_right:Ne \g_@@_pre_code_before_tl
8962   {

```

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`).

```

8963     \@@_subcellcolor:nnnnnnn
8964     {
8965       \tl_if_blank:nTF { #1 }
8966       { { \exp_not:n { #2 } } }
8967       { [ #1 ] { \exp_not:n { #2 } } }
8968     }
8969     { \int_use:N \l_@@_first_row_int } % first row of the block
8970     { \int_use:N \l_@@_first_col_int } % first column of the block
8971     { \int_use:N \l_@@_last_row_int } % last row of the block
8972     { \int_use:N \l_@@_last_col_int } % last column of the block
8973     { \int_use:N \l_@@_split_int }
8974     { \int_use:N \l_@@_split_i_int }
8975   }
8976   \ignorespaces
8977 }

8978 \cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8979 {
8980   \@@_color_opacity: #1
8981   \pgfpicture
8982   \pgf@relevantforpicturesizefalse
8983   \@@_qpoint:n { col - #3 }
8984   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8985   \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8986   \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8987   \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8988   \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8989   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8990   \@@_qpoint:n { row - #2 }
8991   \pgfpathrectanglecorners
8992   { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } \l_@@_tmpc_dim }
8993   { \pgfpoint \l_tmpb_dim \pgf@y }
8994   \pgfusepathqfill
8995   \endpgfpicture
8996 }

```

## 27 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

8997 \keys_define:nn { nicematrix / Auto }
8998 {
8999   columns-type .tl_set:N = \l_@@_columns_type_tl ,
9000   columns-type .value_required:n = true ,
9001   l .meta:n = { columns-type = l } ,
9002   r .meta:n = { columns-type = r } ,
9003   c .meta:n = { columns-type = c } ,
9004   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9005   delimiters / color .value_required:n = true ,
9006   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
9007   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
9008   delimiters .value_required:n = true ,
9009   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
9010   rounded-corners .default:n = 4 pt
9011 }

9012 \NewDocumentCommand \AutoNiceMatrixWithDelims
9013 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
9014 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

9015 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
9016 {

```

The group is for the protection of the keys.

```

9017   \group_begin:
9018   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
9019   \use:e
9020   {
9021     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
9022     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
9023     [ \exp_not:o \l_tmpa_tl ]
9024   }
9025   \int_if_zero:nT \l_@@_first_row_int
9026   {
9027     \int_if_zero:nT \l_@@_first_col_int { & }
9028     \prg_replicate:nn { #4 - 1 } { & }
9029     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9030   }
9031   \prg_replicate:nn { #3 }
9032   {
9033     \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).

```

9034     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
9035     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9036   }
9037   \int_compare:nNnT \l_@@_last_row_int > { -2 }
9038   {
9039     \int_if_zero:nT \l_@@_first_col_int { & }
9040     \prg_replicate:nn { #4 - 1 } { & }
9041     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
9042   }
9043   \end { NiceArrayWithDelims }
9044   \group_end:
9045 }

9046 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
9047 {
9048   \cs_set_protected:cpn { #1 AutoNiceMatrix }

```

```

9049     {
9050         \bool_gset_true:N \g_@@_delims_bool
9051         \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
9052         \AutoNiceMatrixWithDelims { #2 } { #3 }
9053     }
9054 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

9055 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
9056 {
9057     \group_begin:
9058     \bool_gset_false:N \g_@@_delims_bool
9059     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
9060     \group_end:
9061 }

```

## 28 The redefinition of the command `\dotfill`

```

9062 \cs_set_eq:NN \@@_old_dotfill: \dotfill
9063 \cs_new_protected:Npn \@@_dotfill:
9064 {

```

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}`).

```

9065     \@@_old_dotfill:
9066     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
9067 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

9068 \cs_new_protected:Npn \@@_dotfill_i:
9069 {
9070     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim
9071     { \@@_old_dotfill: }
9072 }

```

## 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.

```

9073 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
9074 {
9075     \tl_gput_right:Ne \g_@@_rules_tl
9076     {
9077         \@@_draw_diagbox:nnnnnn
9078         { \int_use:N \c@iRow }
9079         { \int_use:N \c@jCol }
9080         { \int_use:N \c@iRow }
9081         { \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.

```

9082         { \g_@@_row_style_tl \exp_not:n { #1 } }
9083         { \g_@@_row_style_tl \exp_not:n { #2 } }
9084     }

```

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`.

```

9085     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
9086     {
9087         { \int_use:N \c@iRow }
9088         { \int_use:N \c@jCol }
9089         { \int_use:N \c@iRow }
9090         { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

9091     { }
9092 }
9093 }

```

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.

```

9094 \cs_new_protected:Npn \@@_draw_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
9095 {

```

We *must* use an environment `{pgfscope}` and not a simple group of TeX.

```

9096     \begin { pgfscope }
9097     \@@_qpoint:n { row - #1 }
9098     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9099     \@@_qpoint:n { col - #2 }
9100     \dim_set_eq:NN \l_tmpb_dim \pgf@x
9101     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
9102     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
9103     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
9104     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
9105     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
9106     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
9107     {

```

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.

```

9108     \CT@arc@
9109     \pgfsetroundcap
9110     \pgfusepathqstroke
9111 }
9112 \pgfset { inner~sep = 1 pt }
9113 \pgfscope
9114 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
9115 \pgfnode { rectangle } { south~west }
9116 {
9117     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

9118     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
9119     \end { minipage }
9120 }
9121 { }
9122 { }
9123 \endpgfscope
9124 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
9125 \pgfnode { rectangle } { north~east }
9126 {
9127     \begin { minipage } { 20 cm }
9128     \raggedleft
9129     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
9130     \end { minipage }
9131 }
9132 { }

```

```

9133     { }
9134   \end { pgfscope }
9135 }

```

## 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`.

```

9136 \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 `\`.

```

9137 \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`.

```

9138 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
9139 {
9140   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
9141   \@@_CodeAfter_iv:n
9142 }

```

We catch the argument of the command `\end` (in `#1`).

```

9143 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
9144 {

```

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.

```

9145   \str_if_eq:eeTF \@currenvir { #1 }
9146   { \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`.

```

9147   {
9148     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
9149     \@@_CodeAfter_ii:n
9150   }
9151 }

```

## 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.

```

9152 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
9153 {
9154   \pgfpicture
9155   \pgfrememberpicturepositiononpagetrue
9156   \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.

```

9157   \@@_qpoint:n { row - 1 }
9158   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
9159   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
9160   \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).

```

9161   \bool_if:nTF { #3 }
9162   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
9163   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
9164   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
9165   {
9166     \cs_if_exist:cT
9167     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
9168     {
9169       \pgfpointanchor
9170       { \@@_env: - ##1 - #2 }
9171       { \bool_if:nTF { #3 } { west } { east } }
9172       \dim_set:Nn \l_tmpa_dim
9173       {
9174         \bool_if:nTF { #3 }
9175         \dim_min:nn
9176         \dim_max:nn
9177         \l_tmpa_dim
9178         \pgf@x
9179       }
9180     }
9181   }

```

Now we can put the delimiter with a node of PGF.

```

9182   \pgfset { inner~sep = \c_zero_dim }
9183   \dim_zero:N \nulldelimiterspace
9184   \pgftransformshift
9185   {
9186     \pgfpoint
9187     \l_tmpa_dim
9188     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
9189   }
9190   \pgfnode
9191   { rectangle }
9192   { \bool_if:nTF { #3 } { east } { west } }
9193   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

9194   \nullfont
9195   $ % $
9196   \@@_color:o \l_@@_delimiters_color_tl
9197   \bool_if:nTF { #3 } { \left #1 } { \left . }
9198   \vcenter
9199   {
9200     \nullfont
9201     \hrule \@height
9202     \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
9203     \@depth \c_zero_dim
9204     \@width \c_zero_dim
9205   }

```

```

9206     \bool_if:nTF { #3 } { \right . } { \right #1 }
9207     $ % $
9208   }
9209   { }
9210   { }
9211   \endpgfpicture
9212 }

```

## 32 The command \SubMatrix

```

9213 \keys_define:nn { nicematrix / sub-matrix }
9214 {
9215   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
9216   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
9217   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
9218   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
9219   xshift .value_required:n = true ,
9220   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9221   delimiters / color .value_required:n = true ,
9222   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
9223   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9224   hlines .default:n = all ,
9225   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9226   vlines .default:n = all ,
9227   hvlines .meta:n = { hlines, vlines } ,
9228   hvlines .value_forbidden:n = true
9229 }
9230 \keys_define:nn { nicematrix }
9231 {
9232   SubMatrix .inherit:n = nicematrix / sub-matrix ,
9233   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9234   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9235   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
9236 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

9237 \keys_define:nn { nicematrix / SubMatrix }
9238 {
9239   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
9240   delimiters / color .value_required:n = true ,
9241   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
9242   hlines .default:n = all ,
9243   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
9244   vlines .default:n = all ,
9245   hvlines .meta:n = { hlines, vlines } ,
9246   hvlines .value_forbidden:n = true ,
9247   name .code:n =
9248     \tl_if_empty:nTF { #1 }
9249     { \@@_error:n { Invalid-name } }
9250     {
9251       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
9252       {
9253         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
9254         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
9255         {
9256           \str_set:Nn \l_@@_submatrix_name_str { #1 }
9257           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
9258         }
9259       }
9260     }

```



```

9260         { \@@_error:n { Invalid-name } }
9261     } ,
9262     name .value_required:n = true ,
9263     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
9264     rules .value_required:n = true ,
9265     code .tl_set:N = \l_@@_code_tl ,
9266     code .value_required:n = true ,
9267     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
9268 }

9269 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
9270 {
9271     \tl_gput_right:Ne \g_@@_pre_code_after_tl
9272     {
9273         \SubMatrix { #1 } { #2 } { #3 } { #4 }
9274         [
9275             delimiters / color = \l_@@_delimiters_color_tl ,
9276             hlines = \l_@@_submatrix_hlines_clist ,
9277             vlines = \l_@@_submatrix_vlines_clist ,
9278             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
9279             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
9280             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
9281             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
9282             #5
9283         ]
9284     }
9285     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
9286     \ignorespaces
9287 }

9288 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
9289 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9290 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

9291 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
9292 {
9293     \seq_gput_right:Ne \g_@@_submatrix_seq
9294     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9295     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9296     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9297     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9298     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9299 }
9300 }

```

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).

```

9301 \NewDocumentCommand \@@_compute_i_j:nn
9302 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9303 { \@@_compute_i_j:nnnn #1 #2 }

9304 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9305 {
9306     \def \l_@@_first_i_tl { #1 }
9307     \def \l_@@_first_j_tl { #2 }
9308     \def \l_@@_last_i_tl { #3 }
9309     \def \l_@@_last_j_tl { #4 }
9310     \tl_if_eq:NnT \l_@@_first_i_tl { last }
9311     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9312     \tl_if_eq:NnT \l_@@_first_j_tl { last }
9313     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9314     \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

9315      { \tl_set:NV \l_@@_last_i_tl \c{iRow }
9316      \tl_if_eq:NnT \l_@@_last_j_tl { last }
9317      { \tl_set:NV \l_@@_last_j_tl \c{jCol }
9318      }

```

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`.

```

9319 \AtBeginDocument
9320 {
9321   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9322   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9323     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9324 }
9325 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9326 {
9327   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9328   \@@_compute_i_j:nn { #2 } { #3 }
9329   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
9330     { \def \arraystretch { 1 } }
9331   \bool_lazy_or:nnTF
9332     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9333     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9334     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
9335   {
9336     \str_clear_new:N \l_@@_submatrix_name_str
9337     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9338     \pgfpicture
9339     \pgfrememberpicturepositiononpagetrue
9340     \pgf@relevantforpicturesizefalse
9341     \pgfset { inner~sep = \c_zero_dim }
9342     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9343     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9344   \bool_if:NTF \l_@@_submatrix_slim_bool
9345     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
9346     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
9347     {
9348       \cs_if_exist:cT
9349         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9350       {
9351         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9352         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9353           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9354       }
9355       \cs_if_exist:cT

```

```

9356         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9357         {
9358             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9359             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9360             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9361         }
9362     }
9363     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
9364     { \@@_impossible_delimiter:n { left } }
9365     {
9366         \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
9367         { \@@_impossible_delimiter:n { right } }
9368         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9369     }
9370     \endpgfpicture
9371 }
9372 \group_end:
9373 \ignorespaces
9374 }

```

The argument of the following command will be provided by curryfication.

```

9375 \cs_new_protected:Npn \@@_impossible_delimiter:n #1
9376 {
9377     \bool_if:NTF \l_@@_no_cell_nodes_bool
9378     { \@@_error:n { Impossible~SubMatrix~no~cell~nodes } }
9379     { \@@_error:nn { Impossible~SubMatrix } { #1 } }
9380 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9381 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9382 {
9383     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9384     \dim_set:Nn \l_@@_y_initial_dim
9385     {
9386         \fp_to_dim:n
9387         {
9388             \pgf@y
9389             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9390         }
9391     }
9392     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9393     \dim_set:Nn \l_@@_y_final_dim
9394     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9395     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
9396     {
9397         \cs_if_exist:cT
9398         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9399         {
9400             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9401             \dim_set:Nn \l_@@_y_initial_dim
9402             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
9403         }
9404         \cs_if_exist:cT
9405         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9406         {
9407             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9408             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
9409             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9410         }
9411     }
9412     \dim_set:Nn \l_tmpa_dim
9413     {

```

```

9414         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9415         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9416     }
9417     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

9418     \group_begin:
9419     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9420     \@@_set_CTarc:o \l_@@_rules_color_tl
9421     \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`.

```

9422     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9423     {
9424         \int_compare:nNt \l_@@_first_j_tl < { ##1 }
9425         {
9426             \int_compare:nNt
9427             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
9428             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

9429                 \@@_qpoint:n { col - ##1 }
9430                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9431                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9432                 \pgfusepathqstroke
9433             }
9434         }
9435     }

```

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.

```

9436     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
9437     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
9438     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
9439     {
9440         \bool_lazy_and:nnTF
9441         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9442         {
9443             \int_compare_p:nNn
9444             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
9445         {
9446             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
9447             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
9448             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
9449             \pgfusepathqstroke
9450         }
9451         { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
9452     }

```

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.

```

9453     \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
9454     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
9455     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
9456     {
9457         \bool_lazy_and:nnTF
9458         { \int_compare_p:nNn { ##1 } > \c_zero_int }
9459         {
9460             \int_compare_p:nNn
9461             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
9462         {
9463             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

9464     \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
9465     \dim_set:Nn \l_tmpa_dim
9466     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9467     \str_case:nn { #1 }
9468     {
9469         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9470         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9471         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9472         }
9473     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

9474     \dim_set:Nn \l_tmpb_dim
9475     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9476     \str_case:nn { #2 }
9477     {
9478         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9479         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9480         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9481     }
9482     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9483     \pgfusepathqstroke
9484     \group_end:
9485 }
9486 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
9487 }

```

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).

```

9488     \str_if_empty:NF \l_@@_submatrix_name_str
9489     {
9490         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9491         \l_@@_x_initial_dim \l_@@_y_initial_dim
9492         \l_@@_x_final_dim \l_@@_y_final_dim
9493     }
9494     \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`.

```

9495     \begin { pgfscope }
9496     \pgftransformshift
9497     {
9498         \pgfpoint
9499         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9500         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9501     }
9502     \str_if_empty:NTF \l_@@_submatrix_name_str
9503     { \@@_node_left:nn #1 { } }
9504     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9505     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9506     \pgftransformshift
9507     {
9508         \pgfpoint
9509         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9510         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9511     }
9512     \str_if_empty:NTF \l_@@_submatrix_name_str

```

```

9513 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9514 {
9515   \@@_node_right:nnnn #2
9516   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9517 }

```

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`.

```

9518 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9519 \flag_clear_new:N \l_@@_code_flag
9520 \l_@@_code_tl
9521 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ , `row- $i$` , `col- $j$`  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

9522 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of TikZ nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by currying.

```

9523 \cs_new:Npn \@@_pgfpointanchor:n #1
9524 { \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`).

```

9525 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9526 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
9527 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9528 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9529 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

9530 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

9531 { \@@_pgfpointanchor_ii:n { #1 } }
9532 }

```

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`).

```

9533 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9534 { \@@_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`.

```

9535 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

9536 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9537 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

9538 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

9539 { \@@_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`).

```

9540 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9541 }

```

The following function is for the case when the name contains an hyphen.

```

9542 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9543 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9544 \@@_env:
9545 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9546 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9547 }

```

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.

```

9548 \tl_const:Nn \c_@@_integers_alist_tl
9549 {
9550 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9551 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9552 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9553 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9554 }

```

```

9555 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9556 {

```

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`.

```

9557 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9558 {
9559 \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9560 \@@_env: -
9561 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9562 { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9563 { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9564 }
9565 {
9566 \str_if_eq:eeTF { #1 } { last }
9567 {
9568 \flag_raise:N \l_@@_code_flag
9569 \@@_env: -
9570 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9571 { \int_eval:n { \l_@@_last_i_tl + 1 } }
9572 { \int_eval:n { \l_@@_last_j_tl + 1 } }
9573 }

```

```

9574         { #1 }
9575     }
9576 }

```

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`).

```

9577 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9578 {
9579     \pgfnode
9580     { rectangle }
9581     { east }
9582     {
9583         \nullfont
9584         $ % $
9585         \@@_color:o \l_@@_delimiters_color_tl
9586         \left #1
9587         \vcenter
9588         {
9589             \nullfont
9590             \hrule \@height \l_tmpa_dim
9591                 \c_zero_dim
9592                 \c_zero_dim
9593         }
9594         \right .
9595         $ % $
9596     }
9597     { #2 }
9598     { }
9599 }

```

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.

```

9600 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9601 {
9602     \pgfnode
9603     { rectangle }
9604     { west }
9605     {
9606         \nullfont
9607         $ % $
9608         \colorlet { current-color } { . }
9609         \@@_color:o \l_@@_delimiters_color_tl
9610         \left .
9611         \vcenter
9612         {
9613             \nullfont
9614             \hrule \@height \l_tmpa_dim
9615                 \c_zero_dim
9616                 \c_zero_dim
9617         }
9618         \right #1
9619         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9620         ~ { \color { current-color } \smash { #4 } }
9621         $ % $
9622     }
9623     { #2 }
9624     { }
9625 }

```



### 33 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9626 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9627 {
9628   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9629   \ignorespaces
9630 }

9631 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9632 {
9633   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9634   \ignorespaces
9635 }

9636 \keys_define:nn { nicematrix / Brace }
9637 {
9638   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9639   left-shorten .value_forbidden:n = true ,
9640   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9641   right-shorten .value_forbidden:n = true ,
9642   shorten .meta:n = { left-shorten , right-shorten } ,
9643   shorten .value_forbidden:n = true ,
9644   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9645   color .tl_set:N = \l_tmpa_tl ,
9646   color .value_required:n = true ,
9647   unknown .code:n =
9648     \@@_unknown_key:nn
9649     { nicematrix / Brace }
9650     { Unknown~key~for~Brace }
9651 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#2` is the last cell of the rectangle; `#3` is the label of the text; `#4` is the optional argument (a list of *key-value* pairs); `#5` is equal to `under` or `over`.

```

9652 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9653 {
9654   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9655   \@@_compute_i_j:nn { #1 } { #2 }
9656   \bool_lazy_or:nnTF
9657     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9658     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9659     {
9660       \str_if_eq:eeTF { #5 } { under }
9661       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9662       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9663     }
9664   {
9665     \tl_clear:N \l_tmpa_tl
9666     \keys_set:nn { nicematrix / Brace } { #4 }
9667     \tl_if_empty:NF \l_tmpa_tl { \color \l_tmpa_tl }
9668     \pgfpicture
9669     \pgfrememberpicturepositiononpagetrue
9670     \pgf@relevantforpicturesizefalse
9671     \bool_if:NT \l_@@_brace_left_shorten_bool
9672     {
9673       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9674       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9675       {
9676         \cs_if_exist:cT

```

```

9677         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9678         {
9679             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9680
9681             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9682             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9683         }
9684     }
9685 }
9686 \bool_lazy_or:nnT
9687 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9688 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9689 {
9690     \@@_qpoint:n { col - \l_@@_first_j_tl }
9691     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9692 }
9693 \bool_if:NT \l_@@_brace_right_shorten_bool
9694 {
9695     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9696     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9697     {
9698         \cs_if_exist:cT
9699         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9700         {
9701             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9702             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9703             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9704         }
9705     }
9706 }
9707 \bool_lazy_or:nnT
9708 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9709 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9710 {
9711     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9712     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9713 }
9714 \pgfset { inner~sep = \c_zero_dim }
9715 \str_if_eq:eeTF { #5 } { under }
9716 { \@@_underbrace_i:n { #3 } }
9717 { \@@_overbrace_i:n { #3 } }
9718 \endpgfpicture
9719 }
9720 \group_end:
9721 }

```

The argument is the text to put above the brace.

```

9722 \cs_new_protected:Npn \@@_overbrace_i:n #1
9723 {
9724     \@@_qpoint:n { row - \l_@@_first_i_tl }
9725     \pgftransformshift
9726     {
9727         \pgfpoint
9728         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9729         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9730     }
9731     \pgfnode
9732     { rectangle }
9733     { south }
9734     {
9735         \vtop
9736         {
9737             \group_begin:
9738             \everycr { }

```

```

9739         \halign
9740         {
9741             \hfil ## \hfil \crcr
9742             \bool_if:NTF \l_@@_tabular_bool
9743             { \begin { tabular } { c } #1 \end { tabular } }
9744             { $ \begin { array } { c } #1 \end { array } $ }
9745             \cr
9746             $ % $
9747             \overbrace
9748             {
9749                 \hbox_to_wd:nn
9750                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9751                 { }
9752             }
9753             $ % $
9754             \cr
9755             }
9756         \group_end:
9757     }
9758 }
9759 { }
9760 { }
9761 }

```

The argument is the text to put under the brace.

```

9762 \cs_new_protected:Npn \@@_underbrace_i:n #1
9763 {
9764     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9765     \pgftransformshift
9766     {
9767         \pgfpoint
9768         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9769         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9770     }
9771     \pgfnode
9772     { rectangle }
9773     { north }
9774     {
9775         \group_begin:
9776         \everycr { }
9777         \vbox
9778         {
9779             \halign
9780             {
9781                 \hfil ## \hfil \crcr
9782                 $ % $
9783                 \underbrace
9784                 {
9785                     \hbox_to_wd:nn
9786                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9787                     { }
9788                 }
9789                 $ % $
9790                 \cr
9791                 \bool_if:NTF \l_@@_tabular_bool
9792                 { \begin { tabular } { c } #1 \end { tabular } }
9793                 { $ \begin { array } { c } #1 \end { array } $ }
9794                 \cr
9795             }
9796         }
9797         \group_end:
9798     }
9799     { }
9800     { }

```

```
9801 }
```

## 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`).

```
9802 \AddToHook { package / tikz / after }
9803 {
9804   \tikzset
9805   {
9806     nicematrix / brace / .style =
9807     {
9808       decoration = { brace , raise = -0.15 em } ,
9809       decorate ,
9810     } ,
```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```
9811     nicematrix / mirrored-brace / .style =
9812     {
9813       nicematrix / brace ,
9814       decoration = mirror ,
9815     }
9816   }
9817 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```
9818 \keys_define:nn { nicematrix / Hbrace }
9819 {
9820   color .code:n = ,
9821   horizontal-label .code:n = ,
9822   horizontal-labels .code:n = ,
9823   shorten .code:n = ,
9824   shorten-start .code:n = ,
9825   shorten-end .code:n = ,
9826   shorten+ .code:n = ,
9827   shorten-start+ .code:n = ,
9828   shorten-end+ .code:n = ,
9829   shorten~+ .code:n = ,
9830   shorten-start~+ .code:n = ,
9831   shorten-end~+ .code:n = ,
9832   brace-shift .code:n = ,
9833   brace-shift+ .code:n = ,
9834   brace-shift~+ .code:n = ,
9835   unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9836 }
```

Here we need an “fully expandable” command.

```
9837 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9838 {
9839   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9840   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9841   { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9842 }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9843 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9844 {
9845   \int_compare:nNnTF \c@iRow < { 2 }
9846   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9847   \str_if_eq:nnTF { #2 } { * }
9848   {
9849     \bool_set_true:N \l_@@_nullify_dots_bool
9850     \Ldots
9851     [
9852       line-style = nicematrix / brace ,
9853       #1 ,
9854       up =
9855         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9856     ]
9857   }
9858   {
9859     \Hdotsfor
9860     [
9861       line-style = nicematrix / brace ,
9862       #1 ,
9863       up =
9864         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9865     ]
9866     { #2 }
9867   }
9868 }
9869 {
9870   \str_if_eq:nnTF { #2 } { * }
9871   {
9872     \bool_set_true:N \l_@@_nullify_dots_bool
9873     \Ldots
9874     [
9875       line-style = nicematrix / mirrored-brace ,
9876       #1 ,
9877       down =
9878         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9879     ]
9880   }
9881   {
9882     \Hdotsfor
9883     [
9884       line-style = nicematrix / mirrored-brace ,
9885       #1 ,
9886       down =
9887         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9888     ]
9889     { #2 }
9890   }
9891 }
9892 \keys_set:nn { nicematrix / Hbrace } { #1 }
9893 }

9894 \NewDocumentCommand { \@@_Vbrace } { 0 { } } m m {
9895   {
9896     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9897     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9898     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9899   }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not.

```

9900 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9901 {
9902   \int_compare:nNnTF \c@jCol < { 2 }
9903   {
9904     \str_if_eq:nnTF { #2 } { * }
9905     {
9906       \bool_set_true:N \l_@@_nullify_dots_bool
9907       \Vdots
9908       [
9909         Vbrace ,
9910         line-style = nicematrix / mirrored-brace ,
9911         #1 ,
9912         down =
9913         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9914       ]
9915     }
9916     {
9917       \Vdotsfor
9918       [
9919         Vbrace ,
9920         line-style = nicematrix / mirrored-brace ,
9921         #1 ,
9922         down =
9923         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9924       ]
9925       { #2 }
9926     }
9927   }
9928   {
9929     \str_if_eq:nnTF { #2 } { * }
9930     {
9931       \bool_set_true:N \l_@@_nullify_dots_bool
9932       \Vdots
9933       [
9934         Vbrace ,
9935         line-style = nicematrix / brace ,
9936         #1 ,
9937         up =
9938         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9939       ]
9940     }
9941     {
9942       \Vdotsfor
9943       [
9944         Vbrace ,
9945         line-style = nicematrix / brace ,
9946         #1 ,
9947         up =
9948         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9949       ]
9950       { #2 }
9951     }
9952   }
9953   \keys_set:nn { nicematrix / Hbrace } { #1 }
9954 }

```

## 35 The command TikzEveryCell

```

9955 \bool_new:N \l_@@_not_empty_bool
9956 \bool_new:N \l_@@_empty_bool
9957
9958 \keys_define:nn { nicematrix / TikzEveryCell }
9959 {
9960   not-empty .code:n =
9961     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9962     { \bool_set_true:N \l_@@_not_empty_bool }
9963     { \@@_error:n { detection-of-empty-cells } } ,
9964   not-empty .value_forbidden:n = true ,
9965   empty .code:n =
9966     \bool_lazy_or:nnTF \l_@@_in_code_after_bool \g_@@_create_cell_nodes_bool
9967     { \bool_set_true:N \l_@@_empty_bool }
9968     { \@@_error:n { detection-of-empty-cells } } ,
9969   empty .value_forbidden:n = true ,
9970   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9971 }
9972
9973
9974 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9975 {
9976   \IfPackageLoadedTF { tikz }
9977   {
9978     \group_begin:
9979     \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.

```

9980     \tl_set:Nn \l_tmpa_tl { { #2 } }
9981     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9982     { \@@_for_a_block:nnnnn ##1 }
9983     \@@_all_the_cells:
9984     \group_end:
9985   }
9986   { \@@_error:n { TikzEveryCell-without-tikz } }
9987 }
9988
9989
9990 \cs_new_protected:Nn \@@_all_the_cells:
9991 {
9992   \int_step_inline:nn \c@iRow
9993   {
9994     \int_step_inline:nn \c@jCol
9995     {
9996       \cs_if_exist:cF { cell - ##1 - #####1 }
9997       {
9998         \clist_if_in:NcF \l_@@_corners_cells_clist
9999         { ##1 - #####1 }
10000       {
10001         \bool_set_false:N \l_tmpa_bool
10002         \cs_if_exist:cTF
10003         { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
10004         {
10005           \bool_if:NF \l_@@_empty_bool
10006           { \bool_set_true:N \l_tmpa_bool }
10007         }
10008         {
10009           \bool_if:NF \l_@@_not_empty_bool
10010           { \bool_set_true:N \l_tmpa_bool }
10011         }
10012       }

```

```

10013         {
10014             \@@_block_tikz:nnnnn
10015             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
10016         }
10017     }
10018 }
10019 }
10020 }
10021 }
10022
10023 \cs_new_protected:Nn \@@_for_a_block:nnnnn
10024 {
10025     \bool_if:NF \l_@@_empty_bool
10026     {
10027         \@@_block_tikz:nnnnn
10028         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
10029     }
10030     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
10031 }
10032
10033 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
10034 {
10035     \int_step_inline:nnn { #1 } { #3 }
10036     {
10037         \int_step_inline:nnn { #2 } { #4 }
10038         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
10039     }
10040 }

```

## 36 The key draw-trees-in-col

```

10041 \cs_new_protected:Npn \@@_draw_trees:
10042 {
10043     \bool_if:NTF \l_@@_no_cell_nodes_bool
10044     { \@@_error:n { draw-trees-with-no-cell-nodes } }
10045     \@@_draw_trees_i:
10046 }
10047 \cs_new_protected:Npn \@@_draw_trees_i:
10048 {
10049     \@@_expand_clist_hvlines:NN \g_@@_col_with_trees_clist \c@jCol
10050     \dim_zero_new:N \l_@@_em_dim
10051     \dim_set:Nn \l_@@_em_dim { 1 em }
10052     \dim_zero_new:N \l_@@_ex_dim
10053     \dim_set:Nn \l_@@_ex_dim { 1 ex }
10054     \pgfpicture
10055     \pgfrememberpicturepositiononpagetrue
10056     \pgf@relevantforpicturesizefalse
10057     \dim_compare:nNnT \l_@@_trees_line_width_dim > \c_zero_dim
10058     { \pgfsetlinewidth { \l_@@_trees_line_width_dim } }
10059     \@@_color:o \l_@@_trees_color_tl
10060     \pgfsetcornersarced
10061     { \pgfpoint \l_@@_trees_rounded_corners_dim \l_@@_trees_rounded_corners_dim }
10062     \clist_map_function:NN \g_@@_col_with_trees_clist
10063     \@@_draw_trees_in_col:n
10064     \clist_gclear:N \g_@@_col_with_trees_clist
10065     \endpgfpicture
10066 }
10067 \cs_new_protected:Npn \@@_draw_trees_in_col:n #1
10068 {

```

The argument is provided by curryfication.



```

10069 \int_compare:nNnTF { #1 } > \c@jCol
10070 { \@@_error:nn { Col~outside~tabular~in~trees } }
10071 {
10072   \int_compare:nNnTF { #1 } = \c@jCol
10073   { \@@_error:nn { Last~col~in~trees } }
10074   \@@_draw_trees_in_col_i:n
10075 }
10076 { #1 }
10077 }

10078 \cs_new_protected:Npn \@@_draw_trees_in_col_i:n #1
10079 {
10080   \int_set:Nn \l_tmpa_int { 1 }
10081   \int_step_inline:nn { \c@iRow + 1 }
10082   {
10083     \cs_if_exist:cT
10084     { pgf @ sh @ ns @ \@@_env: - ##1 - #1 }
10085     {
10086       \int_compare:nNnT { ##1 } > { \l_tmpa_int + 1 }
10087       {

```

Now, you will, potentially, draw a tree.

```

10088       \@@_draw_tree:nee
10089       { #1 }
10090       { \int_use:N \l_tmpa_int }
10091       { \int_eval:n { ##1 - 1 } }
10092     }
10093     \int_set:Nn \l_tmpa_int { ##1 }
10094   }
10095 }
10096 \int_compare:nNnT \c@iRow > \l_tmpa_int
10097 {
10098   \@@_draw_tree:nee
10099   { #1 }
10100   { \int_use:N \l_tmpa_int }
10101   { \int_eval:n { \c@iRow } }
10102 }
10103 }

```

#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.

```

10104 \cs_new_protected:Npn \@@_draw_tree:nnn #1 #2 #3
10105 {

```

\l\_tmpa\_dim will be the  $x$ -value of the vertical rule that we will draw.

```

10106   \pgfpointanchor { \@@_env: - #1 } { 5 }
10107   \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.

```

10108   \bool_set_false:N \l_tmpa_bool
10109   \int_step_inline:nnnn { #3 } { -1 } { #2 + 1 }
10110   {
10111     \cs_if_exist:cT
10112     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #1 + 1 } }

```

We have found the last branch to draw.

```

10113   {
10114     \pgfpointanchor
10115     { \@@_env: - ##1 - \int_eval:n { #1 + 1 } }
10116     { base~west }
10117     \pgfpathmoveto
10118     {
10119       \pgfpoint
10120       { \dim_eval:n { \pgf@x - 0.7 \l_@@_ex_dim } }

```

```

10121         { \dim_eval:n { \pgf@y + 0.25 \l_@@_em_dim } }
10122     }
10123     \pgfpathlineto { \pgfpoint \l_tmpa_dim \pgf@y }
10124     \bool_if:NF \l_tmpa_bool
10125     {
10126         \pgfpointanchor{ \@@_env: - #2 - #1 } { south }
10127         \pgfpathlineto
10128         { \pgfpoint \l_tmpa_dim { \dim_eval:n { \pgf@y - 3pt } } }
10129         \bool_set_true:N \l_tmpa_bool
10130     }
10131     \pgfusepath { stroke }
10132 }
10133 }
10134 }
10135 \cs_generate_variant:Nn \@@_draw_tree:nnn { n e e }

```

## 37 The key create-blocks-in-col

```

10136 \cs_new_protected:Npn \@@_create_blocks_in_col:
10137 {
10138     \@@_expand_clist_hvlines:NN \g_@@_cbic_clist \c@jCol
10139     \clist_map_inline:Nn \g_@@_cbic_clist
10140     {
10141         \cs_set:cpn
10142         {
10143             pgf @ sh @ ns @ \@@_env:
10144             - \int_eval:n { \c@iRow + 1 } - ##1 }
10145         { rien }

```

\l\_tmpa\_int will be the first row of the block being created.

```

10146         \int_set:Nn \l_tmpa_int { 1 }
10147         \int_step_inline:nn { \c@iRow + 1 }
10148         {
10149             \cs_if_exist:cT
10150             { pgf @ sh @ ns @ \@@_env: - #####1 - ##1 }
10151             {
10152                 \int_compare:nNnT { #####1 } > { \l_tmpa_int + 1 }
10153                 {
10154                     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
10155                     {
10156                         { \int_use:N \l_tmpa_int }
10157                         { ##1 }
10158                         { \int_eval:n { #####1 - 1 } }
10159                         { ##1 }
10160                     }
10161                 }
10162             }
10163             \int_set:Nn \l_tmpa_int { #####1 }
10164         }
10165     }
10166 }
10167 \clist_gclear:N \g_@@_cbic_clist
10168 }

```

## 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.

```

10169 \cs_new_protected:Npn \@@_ShowCellNamesCodeBefore
10170 { \tl_put_right:Nn \@@_actually_color: \@@_ShowCellNames } % noqa

10171 \NewDocumentCommand \@@_ShowCellNames { }
10172 {
10173   \bool_if:NT \l_@@_in_code_after_bool
10174   {
10175     \pgfpicture
10176     \pgfrememberpicturepositiononpagetrue
10177     \pgf@relevantforpicturesizefalse
10178     \pgfpathrectanglecorners
10179     { \@@_qpoint:n { 1 } }
10180     { \@@_qpoint:n { \int_eval:n { 1 + \int_max:nn \c@iRow \c@jCol } } }
10181     \pgfsetfillopacity { 0.75 }
10182     \pgfsetfillcolor { white }
10183     \pgfusepathqfill
10184     \endpgfpicture
10185   }
10186   \dim_gzero_new:N \g_@@_tmpc_dim
10187   \dim_gzero_new:N \g_@@_tmpd_dim
10188   \dim_gzero_new:N \g_@@_tmpe_dim
10189   \int_step_inline:nn \c@iRow
10190   {
10191     \bool_if:NTF \l_@@_in_code_after_bool
10192     {
10193       \pgfpicture
10194       \pgfrememberpicturepositiononpagetrue
10195       \pgf@relevantforpicturesizefalse
10196     }
10197     { \begin { pgfpicture } }
10198     \@@_qpoint:n { row - ##1 }
10199     \dim_set_eq:NN \l_tmpa_dim \pgf@y
10200     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
10201     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
10202     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
10203     \bool_if:NTF \l_@@_in_code_after_bool
10204     { \endpgfpicture }
10205     { \end { pgfpicture } }
10206     \int_step_inline:nn \c@jCol
10207     {
10208       \hbox_set:Nn \l_tmpa_box
10209       {
10210         \normalfont \Large \sffamily \bfseries
10211         \bool_if:NTF \l_@@_in_code_after_bool
10212         { \color { red } }
10213         { \color { red ! 50 } }
10214         ##1 - ####1
10215       }
10216       \bool_if:NTF \l_@@_in_code_after_bool
10217       {
10218         \pgfpicture
10219         \pgfrememberpicturepositiononpagetrue
10220         \pgf@relevantforpicturesizefalse
10221       }
10222       { \begin { pgfpicture } }
10223       \@@_qpoint:n { col - ####1 }
10224       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
10225       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
10226       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
10227       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
10228       \bool_if:NTF \l_@@_in_code_after_bool
10229       { \endpgfpicture }
10230       { \end { pgfpicture } }

```

```

10231 \fp_set:Nn \l_tmpa_fp
10232 {
10233   \fp_min:nn
10234   {
10235     \fp_min:nn
10236     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
10237     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
10238   }
10239   { 1.0 }
10240 }
10241 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
10242 \pgfpicture
10243 \pgfrememberpicturepositiononpagetrue
10244 \pgf@relevantforpicturesizefalse
10245 \pgftransformshift
10246 {
10247   \pgfpoint
10248   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
10249   \g_tmpa_dim
10250 }
10251 \pgfnode
10252 { rectangle }
10253 { center }
10254 { \box_use:N \l_tmpa_box }
10255 { }
10256 { }
10257 \endpgfpicture
10258 }
10259 }
10260 }

```

## 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.

```

10261 \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.

```

10262 \bool_new:N \g_@@_footnote_bool
10263 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
10264 {
10265   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
10266   but-that-key-is-unknown. \l
10267   It-will-be-ignored. \l
10268   For-a-list-of-the-available-keys,~type-H~<return>.
10269 }
10270 {
10271   The-available-keys-are~(in-alphabetic-order):~
10272   footnote,~
10273   footnotehyper,~
10274   messages-for-Overleaf,~
10275   renew-dots-and~

```

```

10276     renew-matrix.
10277 }
10278 \keys_define:nn { nicematrix }
10279 {
10280     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
10281     renew-dots .value_forbidden:n = true ,
10282     renew-matrix .code:n = \@@_renew_matrix: ,
10283     renew-matrix .value_forbidden:n = true ,
10284     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
10285     footnote .bool_set:N = \g_@@_footnote_bool ,
10286     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
10287     unknown .code:n = \@@_error:n { Unknown~key~for~package }
10288 }
10289 \ProcessKeyOptions

10290 \@@_msg_new:nn { footnote-with-footnotehyper-package }
10291 {
10292     You~can't~use~the~option~'footnote'~because~the~package~
10293     footnotehyper~has~already~been~loaded.~
10294     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
10295     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10296     of~the~package~footnotehyper.\\
10297     The~package~footnote~won't~be~loaded.
10298 }
10299 \@@_msg_new:nn { footnotehyper-with-footnote-package }
10300 {
10301     You~can't~use~the~option~'footnotehyper'~because~the~package~
10302     footnote~has~already~been~loaded.~
10303     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
10304     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
10305     of~the~package~footnote.\\
10306     The~package~footnotehyper~won't~be~loaded.
10307 }

10308 \bool_if:NT \g_@@_footnote_bool
10309 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10310     \IfClassLoadedTF { beamer }
10311     { \bool_set_false:N \g_@@_footnote_bool }
10312     {
10313         \IfPackageLoadedTF { footnotehyper }
10314         { \@@_error:n { footnote-with-footnotehyper~package } }
10315         { \usepackage { footnote } }
10316     }
10317 }

10318 \bool_if:NT \g_@@_footnotehyper_bool
10319 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

10320     \IfClassLoadedTF { beamer }
10321     { \bool_set_false:N \g_@@_footnote_bool }
10322     {
10323         \IfPackageLoadedTF { footnote }
10324         { \@@_error:n { footnotehyper-with-footnote~package } }
10325         { \usepackage { footnotehyper } }
10326     }
10327     \bool_set_true:N \g_@@_footnote_bool
10328 }

```

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.

```

10329 \bool_new:N \l_@@_underscore_loaded_bool
10330 \IfPackageLoadedT { underscore }
10331   { \bool_set_true:N \l_@@_underscore_loaded_bool }
10332 \AtBeginDocument
10333   {
10334     \bool_if:NF \l_@@_underscore_loaded_bool
10335     {
10336       \IfPackageLoadedT { underscore }
10337         { \@@_error:n { underscore~after~nicematrix } }
10338     }
10339   }

```

## 41 Compatibility with threeparttable

```

10340 \AtBeginDocument
10341   {
10342     \IfPackageLoadedT { threeparttable }
10343     {
10344       \AddToHook { env / threeparttable / begin }
10345       {
10346         \TPT@hookin { NiceTabular }
10347         \TPT@hookin { NiceTabular* }
10348         \TPT@hookin { NiceTabularX }
10349       }
10350     }
10351   }

```

## 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.

```

10352 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
10353   {
10354     \str_if_eq:eeTF
10355       { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
10356       { + }
10357       {
10358         \str_set:Ne \l_tmpa_str
10359           { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
10360         \bool_set_false:N \l_tmpa_bool
10361         \clist_map_inline:nn { #1 }
10362           {

```

```

10363         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10364         {
10365             \@@_error:n { key~without~+~exists }
10366             \bool_set_true:N \l_tmpa_bool
10367             \clist_map_break:
10368         }
10369     }
10370     \bool_if:NF \l_tmpa_bool
10371     {
10372         \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
10373         \@@_unknown_key_i:nn { #1 } { #2 }
10374     }
10375 }
10376 { \@@_unknown_key_i:nn { #1 } { #2 } }
10377 }

```

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.

```

10378 \cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
10379 {
10380     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
10381     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
10382     \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
10383     \bool_set_false:N \l_tmpa_bool
10384     \clist_map_inline:nn { #1 }
10385     {
10386         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
10387         {
10388             \@@_error:n { key~with~normal~form~exists }
10389             \bool_set_true:N \l_tmpa_bool
10390             \clist_map_break:
10391         }
10392     }
10393     \bool_if:NF \l_tmpa_bool
10394     {
10395         \@@_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).

```

10396         \bool_if:NF \g_@@_messages_for_Overleaf_bool
10397         { \msg_info:nn { nicematrix } { #2~+ } }
10398     }
10399 }

10400 \@@_msg_new:nn { key~without~+~exists }
10401 {
10402     The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
10403     additive~syntax~(with~+=)~.~It~will~be~ignored.
10404 }

10405 \@@_msg_new:nn { key~with~normal~form~exists }
10406 {
10407     No~key~'\l_keys_key_str'.\\
10408     It~will~be~ignored.~Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
10409 }

10410 \str_const:Ne \c_@@_available_keys_str
10411 {
10412     \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }

```

```

10413     { For-a-list-of-the-available-keys,~type-H~<return>. }
10414 }
10415 \seq_new:N \g_@@_types_of_matrix_seq
10416 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
10417 {
10418     NiceMatrix ,
10419     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
10420 }
10421 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
10422 { \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`.

```

10423 \cs_new_protected:Npn \@@_err_too_many_cols:
10424 {
10425     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
10426     { \@@_fatal:nn { too-many-cols-for-array } }
10427     \int_compare:nNnT \l_@@_last_col_int = { -2 }
10428     { \@@_fatal:n { too-many-cols-for-matrix } }
10429     \int_compare:nNnT \l_@@_last_col_int = { -1 }
10430     { \@@_fatal:n { too-many-cols-for-matrix } }
10431     \bool_if:NF \l_@@_last_col_without_value_bool
10432     { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
10433 }

```

The following command must *not* be protected since it's used in an error message.

```

10434 \cs_new:Npn \@@_message_hdotsfor:
10435 {
10436     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
10437     { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ or~
10438       \token_to_str:N \Hbrace \ is-incorrect. }
10439 }
10440 \cs_new_protected:Npn \@@_Hline_in_cell:
10441 { \@@_error:n { Misuse-of-Hline } }
10442 \@@_msg_new:nn { Misuse-of-Hline }
10443 {
10444     Misuse-of~\token_to_str:N \Hline. \\
10445     Error-in-the-row~ \int_use:N \c@iRow\ of-your~\@@_full_name_env:.~
10446     \token_to_str:N \Hline\ (like~\token_to_str:N \hline)~must-be-used-only~
10447     at-the-beginning-of-a-row.~Your-command-will-be-ignored.
10448 }
10449 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
10450 {
10451     Incompatible-options.\\
10452     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-the-same-time.~
10453     The-output-will-not-be-reliable.
10454 }
10455 \@@_msg_new:nn { Body-alone }
10456 {
10457     \token_to_str:N \Body\ alone. \\
10458     You-have-used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.
10459 }
10460 \@@_msg_new:nn { Bad-use-of~CodeBefore }
10461 {
10462     Bad-use-of~\token_to_str:N \CodeBefore. \\
10463     \token_to_str:N \CodeBefore\ must-be-used-only-at-the-beginning-of~
10464     the-environment.
10465 }

```



```

10466 \@@_msg_new:nn { NiceTabularX-probably-required }
10467 {
10468   Incorrect-syntax.\
10469   You-probably-want-to-use-\{NiceTabularX\}-whose-first-argument-is-the~
10470   ~width-that-you-wish-for-your-tabular.~But-you-can-also-use-\{NiceTabular\}~
10471   with-the-key-'width'.
10472 }
10473 \@@_msg_new:nn { cellcolor-in-Block }
10474 {
10475   Bad-use-of-\token_to_str:N \cellcolor \
10476   You-can't-use-\token_to_str:N \cellcolor\ in-\token_to_str:N \Block\
10477   \bool_if:NTF \l_@@_amp_in_blocks_bool
10478   { (but-you-could-use-it-in-a-sub-block-since-'&-in-blocks'-is-in-force) }
10479   { (it's-possible-in-a-sub-block-when-'&-in-blocks'-is-in-force) }
10480   .~Here,~you-should-use-the-key-'fill'-of-the-block.~Your-command-will-be-ignored.
10481 }
10482 \@@_msg_new:nn { rowcolor-in-Block }
10483 {
10484   Bad-use-of-\token_to_str:N \rowcolor \
10485   You-can't-use-\token_to_str:N \rowcolor\ in-\token_to_str:N \Block.~
10486   However,~it's-possible-to-color-the-block-with-its-key-'fill'.~
10487   Your-command-will-be-ignored.
10488 }
10489 \@@_msg_new:nn { key-color-inside }
10490 {
10491   Deleted-key.\
10492   The-key-'color-inside'~(and-its-alias-'colortbl-like')~has-been-deleted-in
10493   ~'nicematrix'~and-must-not-be-used.
10494 }
10495 \@@_msg_new:nn { Invalid-argument-for-w }
10496 {
10497   Invalid-argument-for-w.\
10498   You-have-used-the-type-of-alignment-~'#1'~for-your-column-'w'~
10499   but-only-'c',~'r',~'l'~and-'s'~are-allowed-in-a-column-'w'.~
10500   If-you-go-on,~'c'~will-be-used.
10501 }
10502 \@@_msg_new:nn { invalid-weight }
10503 {
10504   Unknown-key.\
10505   The-key~'\l_keys_key_str'~of-your-column-X-is-unknown-and-will-be-ignored.~
10506   The-available-keys-are:~l,~c,~r,~t(=p),~m,~b,~V~
10507   \IfPackageLoadedTF { varwidth }
10508   { (since-'varwidth'-is-loaded)~}
10509   { (if-you-load-'varwidth')~}
10510   and-real-numbers-for-the-weight-of-the-X-column.
10511 }
10512 \@@_msg_new:nn { last-col-not-used }
10513 {
10514   Column-not-used.\
10515   The-key-'last-col'~is-in-force-but-you-have-not-used-that-last-column~
10516   in-your~\@@_full_name_env: .~However,~you-can-go-on.
10517 }
10518 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10519 {
10520   Too-many-columns.\
10521   In-the-row~\int_eval:n { \c@iRow },~
10522   you-try-to-use-more-columns~
10523   than-allowed-by-your~\@@_full_name_env: .
10524   \@@_message_hdotsfor: \
10525   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
10526   (plus-the-exterior-columns).~

```

```

10527 But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10528 }
10529 \@@_msg_new:nn { too-many-cols-for-matrix }
10530 {
10531   Too-many-columns.\.
10532   In-the-row~ \int_eval:n { \c@iRow } ,~
10533   you-try-to-use-more-columns-than-allowed-by-your~ \@@_full_name_env: .
10534   \@@_message_hdotsfor: \
10535   Recall-that-the-maximal-number-of-columns-for-a-matrix~
10536   (excepted-the-potential-exterior-columns)-is-fixed-by-the~
10537   LaTeX-counter~'MaxMatrixCols'.~
10538   Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
10539   (use~ \token_to_str:N \setcounter \ to-change-that-value).~
10540   But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10541 }
10542 \@@_msg_new:nn { too-many-cols-for-array }
10543 {
10544   Too-many-columns.\.
10545   In-the-row~ \int_eval:n { \c@iRow } ,~
10546   ~you-try-to-use-more-columns-than-allowed-by-your~
10547   \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
10548   \int_use:N \g_@@_static_num_of_col_int \
10549   \bool_if:nT
10550     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10551     { (plus-the-exterior-ones)~}
10552   since-the-preamble-is~' \g_@@_user_preamble_tl '~
10553   But,~maybe,~you-have-forgotten-a-\token_to_str:N \\.
10554 }
10555 \@@_msg_new:nn { columns-not-used }
10556 {
10557   Columns-not-used.\.
10558   The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10559   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
10560   columns-but-you-only-used~ \int_use:N \c@jCol .~The-columns-you-did-not~
10561   used-won't-be-created.~You-won't-have-similar-warning-till-the-end-of-the-document.
10562 }
10563 }
10564 \@@_msg_new:nn { Bad-use-of-NiceTabularNotes }
10565 {
10566   Bad-use-of-\token_to_str:N \NiceTabularNotes \.
10567   \token_to_str:N~\NiceTabularNotes\ should-be-used-only-after-at-tabular~
10568   which-uses~notes/no-print`.~ Your-command-will-be-ignored.
10569 }
10570 \@@_msg_new:nn { empty-preamble }
10571 {
10572   Empty-preamble.\.
10573   The-preamble-of-your~ \@@_full_name_env: \ is-empty.
10574 }
10575 \@@_msg_new:nn { in-first-col }
10576 {
10577   Erroneous-use.\.
10578   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.~
10579   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'first-col'.
10580 }
10581 \@@_msg_new:nn { in-last-col }
10582 {
10583   Erroneous-use.\.
10584   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.~
10585   Your-command-will-be-ignored.~You-can-try-to-delete-the-key~'last-col'.
10586 }

```

```

10587 \@@_msg_new:nn { in~first~row }
10588 {
10589   Erroneous~use.\\
10590   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.~
10591   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'first~row'.
10592 }
10593 \@@_msg_new:nn { in~last~row }
10594 {
10595   Erroneous~use.\\
10596   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.~
10597   Your~command~will~be~ignored.~You~can~try~to~delete~the~key~'last~row'.
10598 }
10599 \@@_msg_new:nn { TopRule~without~booktabs }
10600 {
10601   Erroneous~use.\\
10602   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.~
10603   You~should~load~'booktabs'~(before~or~after~'nicematrix').~
10604   Your~command~will~be~ignored.
10605 }
10606 \@@_msg_new:nn{ rotate~in~p~col }
10607 {
10608   \token_to_str:N \rotate\ forbidden.\\
10609   You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
10610   'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
10611   If~you~go~on,~maybe~you~won't~have~the~expected~output.~You~should~
10612   consider~the~classical~command~\token_to_str:N \rotatebox.
10613 }
10614 \@@_msg_new:nn { TopRule~without~tikz }
10615 {
10616   Erroneous~use.\\
10617   You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.~
10618   You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
10619   \IfPackageLoadedF { booktabs }
10620     { You~should~also~load~'booktabs'~
10621       with~\token_to_str:N \usepackage \{booktabs\}.~ }
10622   Your~command~will~be~ignored.
10623 }
10624 \@@_msg_new:nn { caption~outside~float }
10625 {
10626   Key~caption~forbidden.\\
10627   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10628   environment~(such~as~\{table\}).~Your~key~will~be~ignored.
10629 }
10630 \@@_msg_new:nn { short~caption~without~caption }
10631 {
10632   You~should~not~use~the~key~'short~caption'~without~'caption'.~
10633   However,~your~'short~caption'~will~be~used~as~'caption'.
10634 }
10635 \@@_msg_new:nn { double~closing~delimiter }
10636 {
10637   Double~delimiter.\\
10638   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10639   delimiter.~This~delimiter~will~be~ignored.~You~can~try~to~use~
10640   \token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
10641 }
10642 \@@_msg_new:nn { delimiter~after~opening }
10643 {
10644   Double~delimiter.\\
10645   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10646   delimiter.~That~delimiter~will~be~ignored.
10647 }

```

```

10648 \@@_msg_new:nn { bad-option-for-line-style }
10649 {
10650   Bad-line-style.\
10651   Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line-style'~
10652   is~'standard'.~Your~key~will~be~ignored.~You~can~load~TikZ~with~
10653   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.
10654 }
10655 \@@_msg_new:nn { draw-trees-with-no-cell-nodes }
10656 {
10657   Incompatible-keys.\
10658   You~can't~use~the~key~'draw-trees-in-col'~here~because~the~key~'no-cell-nodes'~
10659   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10660   is~to~speed~compilation~up).~If~you~go~on,~that~key~will~be~ignored.
10661 }
10662 \@@_msg_new:nn { corners-with-no-cell-nodes }
10663 {
10664   Incompatible-keys.\
10665   You~can't~use~the~key~'corners'~here~because~the~key~'no-cell-nodes'~
10666   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10667   is~to~speed~compilation~up).\
10668   If~you~go~on,~that~key~will~be~ignored.
10669 }
10670 \@@_msg_new:nn { extra-nodes-with-no-cell-nodes }
10671 {
10672   Incompatible-keys.\
10673   You~can't~create~'extra-nodes'~here~because~the~key~'no-cell-nodes'~
10674   is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
10675   is~to~speed~compilation~up).~If~you~go~on,~those~extra~nodes~won't~be~created.
10676 }
10677 \@@_msg_new:nn { Identical-notes-in-caption }
10678 {
10679   Identical~tabular~notes.\
10680   You~can't~put~several~notes~with~the~same~content~in~
10681   \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).~
10682   If~you~go~on,~the~output~will~probably~be~erroneous.
10683 }
10684 \@@_msg_new:nn { tabularnote~below~the~tabular }
10685 {
10686   \token_to_str:N \tabularnote \ forbidden\
10687   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10688   of~your~tabular~because~the~caption~will~be~composed~below~
10689   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
10690   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .~
10691   Your~ \token_to_str:N \tabularnote \ will~be~ignored~and~
10692   no~similar~error~will~raised~in~this~document.
10693 }
10694 \@@_msg_new:nn { Unknown-key~for~rules }
10695 {
10696   Unknown~key.\
10697   There~are~only~three~keys~available~here:~'width',~'color'~and~
10698   'fix-vertex'.~Your~key~' \l_keys_key_str ' will~be~ignored.
10699 }
10700 \@@_msg_new:nn { Unknown-key~for~trees }
10701 {
10702   Unknown~key.\
10703   There~are~only~three~keys~available~here:~width~color~and~
10704   rounded-corners.~Your~key~' \l_keys_key_str ' will~be~ignored.
10705 }
10706 % \end{macrocode}
10707 %
10708 %

```

```

10709 % \begin{macrocode}
10710 \@@_msg_new:nn { Unknown~key~for~Hbrace }
10711 {
10712   Unknown~key.\\
10713   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10714   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10715   and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
10716   'horizontal-label(s)',~'shorten'~'shorten-end'~
10717   and~'shorten-start'.
10718 }
10719 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10720 {
10721   Unknown~key.\\
10722   There~is~only~two~keys~available~here:~
10723   'empty'~and~'not-empty'.~Your~key~' \l_keys_key_str '~will~be~ignored.
10724 }
10725 \@@_msg_new:nn { Unknown~key~for~rotate }
10726 {
10727   Unknown~key.\\
10728   The~only~keys~available~here~are~'c'~and~'-90'.~
10729   Your~key~' \l_keys_key_str '~will~be~ignored.
10730 }
10731 \@@_msg_new:nnn { Unknown~key~for~custom~line }
10732 {
10733   Unknown~key.\\
10734   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
10735   It~you~go~on,~you~will~probably~have~other~errors. \\
10736   \c_@@_available_keys_str
10737 }
10738 {
10739   The~available~keys~are~(in~alphabetic~order):~
10740   ccommand,~
10741   color,~
10742   command,~
10743   dotted,~
10744   letter,~
10745   multiplicity,~
10746   sep-color,~
10747   tikz,~and~total~width.
10748 }
10749 \@@_msg_new:nnn { Unknown~key~for~default~line }
10750 {
10751   Unknown~key.\\
10752   The~key~' \l_keys_key_str '~is~unknown~in~a~'default~line'.~
10753   It~you~go~on,~you~will~probably~have~other~errors. \\
10754   \c_@@_available_keys_str
10755 }
10756 {
10757   The~available~keys~are~(in~alphabetic~order):~
10758   color,~
10759   dotted,~
10760   multiplicity,~
10761   sep-color,~
10762   tikz,~and~total~width.
10763 }
10764 \@@_msg_new:nnn { Unknown~key~for~xdots }
10765 {
10766   Unknown~key.\\
10767   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10768   \c_@@_available_keys_str
10769 }
10770 {

```

```

10771 The~available~keys~are~(in~alphabetic~order):~
10772 'color',~
10773 'horizontal(s)-labels',~
10774 'inter',~
10775 'line-style',~
10776 'nullify',~
10777 'radius',~
10778 'shorten',~
10779 'shorten-end'~and~'shorten-start'.
10780 }

10781 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10782 {
10783   Unknown~key.\\
10784   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10785   (and~you~try~to~use~' \l_keys_key_str ').~Your~key~will~be~ignored.
10786 }

10787 \@@_msg_new:nn { Col~outside~tabular~in~trees }
10788 {
10789   Error~with~'draw-trees-in-col' \\
10790   The~number~of~column~'#1'~is~outside~your~tabular~since~the~last~column~
10791   is~\int_use:N \c@jCol.~It~will~be~ignored.
10792 }

10793 \@@_msg_new:nn { Last~col~in~trees }
10794 {
10795   Error~with~'draw-trees-in-col' \\
10796   You~can't~use~'draw-trees-in-col'~with~the~column~'#1'~ because~it's~
10797   the~last~column~of~your~tabular.~It~will~be~ignored.
10798 }

10799 \@@_msg_new:nn { label~without~caption }
10800 {
10801   You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10802   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10803 }

10804 \@@_msg_new:nn { W~warning }
10805 {
10806   Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10807   (row~ \int_use:N \c@iRow ).
10808 }

10809 \@@_msg_new:nn { Construct~too~large }
10810 {
10811   Construct~too~large.\\
10812   Your~command~ \token_to_str:N #1
10813   can't~be~drawn~because~your~matrix~is~too~small.~
10814   Your~command~will~be~ignored.
10815 }

10816 \@@_msg_new:nn { underscore~after~nicematrix }
10817 {
10818   Problem~with~'underscore'.\\
10819   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10820   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10821   ' \token_to_str:N \Cdots \token_to_str:N _
10822   \{ n \token_to_str:N \text \{ ~times \} \}' .
10823 }

10824 \@@_msg_new:nn { ampersand~in~light~syntax }
10825 {
10826   Ampersand~forbidden.\\
10827   You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10828   ~the~key~'light-syntax'~is~in~force.
10829 }

10830 \@@_msg_new:nn { double~backslash~in~light~syntax }

```

```

10831 {
10832   Double~backslash~forbidden.\\
10833   You~can't~use~ \token_to_str:N \\
10834   ~to~separate~rows~because~the~key~'light-syntax'~
10835   is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10836   (set~by~the~key~'end-of-row').
10837 }
10838 \@@_msg_new:nn { bad~value~for~baseline }
10839 {
10840   Bad~value~for~baseline.\\
10841   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10842   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
10843   \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
10844   the~form~'line-i'.~A~value~of~1~will~be~used.
10845 }
10846 \@@_msg_new:nn { bad~value~for~baseline~line }
10847 {
10848   Bad~value~for~baseline~with~line.\\
10849   The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
10850   valid.~The~number~of~the~line~must~be~between~1~and~
10851   \int_eval:n { \c@iRow + 1 }.~
10852   A~value~of~'line-1'~will~be~used.
10853 }
10854 \@@_msg_new:nn { detection~of~empty~cells }
10855 {
10856   Problem~with~'not~empty'\\
10857   For~technical~reasons,~you~must~activate~
10858   'create~cell~nodes'~in~ \token_to_str:N \CodeBefore \
10859   in~order~to~use~the~key~' \l_keys_key_str '~
10860   Your~key~will~be~ignored.
10861 }
10862 \@@_msg_new:nn { siunitx~too~old }
10863 {
10864   siunitx~too~old\\
10865   You~can't~use~the~columns~'S'~because~your~version~of~'siunitx'~
10866   is~too~old.~You~need~at~least~the~version~3.5.1~(2026/03/26).
10867 }
10868 \@@_msg_new:nn { Invalid~name }
10869 {
10870   Invalid~name.\\
10871   You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10872   \SubMatrix \ of~your~ \@@_full_name_env: .~
10873   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
10874   Your~key~will~be~ignored.
10875 }
10876 \@@_msg_new:nn { Hbrace~not~allowed }
10877 {
10878   Command~not~allowed.\\
10879   You~can't~use~the~command~ \token_to_str:N #1
10880   because~you~have~not~loaded~
10881   \IfPackageLoadedTF { tikz }
10882   { the~TikZ~library~'decorations.pathreplacing'.~Use~ }
10883   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10884   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10885   Your~command~will~be~ignored.
10886 }
10887 \@@_msg_new:nn { Vbrace~not~allowed }
10888 {
10889   Command~not~allowed.\\
10890   You~can't~use~the~command~ \token_to_str:N \Vbrace \
10891   because~you~have~not~loaded~TikZ~

```

```

10892     and-the-TikZ-library~'decorations.pathreplacing'.~
10893     Use: ~\token_to_str:N \usepackage \{tikz\}~
10894     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10895     Your~command~will~be~ignored.
10896 }
10897 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10898 {
10899     Wrong~line.\\
10900     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10901     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10902     number~is~not~valid.~It~will~be~ignored.
10903 }
10904 \@@_msg_new:nn { Impossible~SubMatrix }
10905 {
10906     Impossible~SubMatrix.\\
10907     It's~impossible~to~draw~your~\token_to_str:N \SubMatrix \
10908     because~all~the~cells~are~empty~in~the~column~on~the~#1~
10909     side~of~that~\token_to_str:N \SubMatrix.
10910     \bool_if:NT \l_@@_submatrix_slim_bool
10911     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10912     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10913 }
10914 \@@_msg_new:nn { Impossible~SubMatrix~no~cell~nodes }
10915 {
10916     Impossible~SubMatrix.\\
10917     It's~impossible~to~draw~your~ \token_to_str:N \SubMatrix \
10918     because~the~key~'no~cell~nodes'~is~in~force.~
10919     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10920 }
10921 \@@_msg_new:nnn { width~without~X~columns }
10922 {
10923     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10924     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .~
10925     Your~key~will~be~ignored.
10926 }
10927 {
10928     This~message~is~the~message~'width~without~X~columns'~
10929     of~the~module~'nicematrix'.~
10930     The~experimented~users~can~disable~that~message~with~
10931     \token_to_str:N \msg_redirect_name:nnn .\\
10932 }
10933
10934 \@@_msg_new:nn { key~multiplicity~with~dotted }
10935 {
10936     Incompatible~keys. \\
10937     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10938     in~a~'custom~line'.~They~are~incompatible.~
10939     The~key~'multiplicity'~will~be~ignored.
10940 }
10941 \@@_msg_new:nn { empty~environment }
10942 {
10943     Empty~environment.\\
10944     Your~ \@@_full_name_env: \ is~empty.
10945 }
10946 \@@_msg_new:nn { No~letter~and~no~command }
10947 {
10948     Erroneous~use.\\
10949     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10950     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10951     '~ccommand'~(to~draw~horizontal~rules).~However,~you~can~go~on.
10952 }

```



```

10953 \@@_msg_new:nn { Forbidden~letter }
10954 {
10955   Forbidden~letter.\\
10956   You~can't~use~the~letter~'#1'~for~a~customized~line.~
10957   It~will~be~ignored.~The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10958 }
10959 \@@_msg_new:nn { Several~letters }
10960 {
10961   Wrong~name.\\
10962   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10963   have~used~'\l_@@_letter_str '~).~It~will~be~ignored.
10964 }
10965 \@@_msg_new:nn { Delimiter~with~small }
10966 {
10967   Delimiter~forbidden.\\
10968   You~can't~put~a~delimiter~in~the~preamble~of~your~
10969   \@@_full_name_env: \
10970   because~the~key~'small'~is~in~force.
10971 }
10972 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10973 {
10974   Unknown~cell.\\
10975   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10976   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10977   can't~be~executed~because~a~cell~doesn't~exist.~
10978   Your~command~ \token_to_str:N \line \ will~be~ignored.
10979 }
10980 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10981 {
10982   Duplicate~name. \\
10983   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10984   in~this~ \@@_full_name_env: .~Your~key~will~be~ignored.~
10985   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10986   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10987 }
10988 {
10989   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10990   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10991 }
10992 \@@_msg_new:nn { r~or~l~with~preamble }
10993 {
10994   Erroneous~use. \\
10995   You~can't~use~the~key~'\l_keys_key_str '~in~your~ \@@_full_name_env: .~
10996   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10997   your~ \@@_full_name_env: .~
10998   Your~key~will~be~ignored.
10999 }
11000 \@@_msg_new:nn { Hdotsfor~in~col~0 }
11001 {
11002   Erroneous~use. \\
11003   You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
11004   in~an~exterior~column~of~the~array.
11005 }
11006 \@@_msg_new:nn { bad~corner }
11007 {
11008   Bad~corner. \\
11009   '#1'~is~an~incorrect~specification~for~a~corner~(in~the~key~
11010   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.~
11011   This~specification~of~corner~will~be~ignored.
11012 }

```

```

11013 \@@_msg_new:nn { bad-border }
11014 {
11015   Bad-border.\\
11016   '\l_keys_key_str'~is~an~incorrect~specification~for~a~border~
11017   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
11018   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
11019   also~use~the~key~'tikz'
11020   \IfPackageLoadedF { tikz }
11021   { ~if~you~load~the~LaTeX~package~'tikz' } ).~
11022   This~specification~of~border~will~be~ignored.
11023 }
11024 \@@_msg_new:nn { TikzEveryCell~without~tikz }
11025 {
11026   TikZ~not~loaded. \\
11027   You~can't~use~ \token_to_str:N \TikzEveryCell \
11028   because~you~have~not~loaded~TikZ.~You~can~load~TikZ~with~
11029   \token_to_str:N \usepackage \{tikz\},~before~or~after~'nicematrix'.~
11030   Your~command~will~be~ignored.
11031 }
11032 \@@_msg_new:nn { tikz~key~without~tikz }
11033 {
11034   TikZ~not~loaded. \\
11035   You~can't~use~the~key~'tikz'~for~the~command~ \token_to_str:N
11036   \Block '~because~you~have~not~loaded~TikZ.~
11037   You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\},~
11038   before~or~after~'nicematrix'.~Your~key~will~be~ignored.
11039 }
11040 \@@_msg_new:nn { Bad~argument~for~Block }
11041 {
11042   Bad~argument. \\
11043   The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
11044   be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
11045   '#1'.~ If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
11046   the~argument~was~empty).
11047 }
11048 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
11049 {
11050   Erroneous~use. \\
11051   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
11052   'last~col'~without~value.~However,~you~can~go~on~for~this~time~
11053   (the~value~' \l_keys_value_tl '~will~be~ignored).
11054 }
11055 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
11056 {
11057   Erroneous~use. \\
11058   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
11059   'last~col'~without~value.~ However,~you~can~go~on~for~this~time~
11060   (the~value~' \l_keys_value_tl '~will~be~ignored).
11061 }
11062 \@@_msg_new:nn { Block~too~large~1 }
11063 {
11064   Block~too~large. \\
11065   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
11066   too~small~for~that~block.~This~block~and~maybe~others~will~be~ignored.
11067 }
11068 \@@_msg_new:nn { Block~too~large~2 }
11069 {
11070   Block~too~large. \\
11071   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
11072   \g_@@_static_num_of_col_int \
11073   columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~

```

```

11074     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
11075     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: .~
11076     This-block-and-maybe-others-will-be-ignored.
11077 }
11078 \@@_msg_new:nn { unknown~column-type }
11079 {
11080     Bad-column-type. \\\
11081     The-column-type~'#1'~in-your~ \@@_full_name_env: \ is-unknown.
11082 }
11083 \@@_msg_new:nn { unknown~column-type-multicolumn }
11084 {
11085     Bad-column-type. \\\
11086     The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
11087     ~of-your~ \@@_full_name_env: \ is-unknown.
11088 }
11089 \@@_msg_new:nn { unknown~column-type-S }
11090 {
11091     Bad-column-type. \\\
11092     The-column-type~'S'~in-your~ \@@_full_name_env: \ is-unknown.~
11093     If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
11094     load-that-package-with~\token_to_str:N \usepackage \{siunitx\}.
11095 }
11096 \@@_msg_new:nn { unknown~column-type-S-multicolumn }
11097 {
11098     Bad-column-type. \\\
11099     The-column-type~'S'~in-the-command~\token_to_str:N \multicolumn \
11100     of-your~ \@@_full_name_env: \ is-unknown.~If-you-want-to-use-the-column~
11101     type~'S'~of~siunitx,~you-should-load-that-package-with~
11102     \token_to_str:N \usepackage \{siunitx\}.
11103 }
11104 \@@_msg_new:nn { tabularnote~forbidden }
11105 {
11106     Forbidden-command. \\\
11107     You-can't-use-the-command~ \token_to_str:N \tabularnote \
11108     ~here.~This-command-is-available-only-in~
11109     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in~
11110     the-argument-of-a-command~\token_to_str:N \caption \ included~
11111     in-an-environment~\{table\}.~Your-command-will-be-ignored.
11112 }
11113 \@@_msg_new:nn { borders~forbidden }
11114 {
11115     Forbidden-key.\\
11116     You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
11117     because-the-option~'rounded-corners'~is-in-force-with-a-non-zero-value.~
11118     Your-key-will-be-ignored.
11119 }
11120 \@@_msg_new:nn { bottomrule~without~booktabs }
11121 {
11122     booktabs-not-loaded.\\
11123     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
11124     loaded~'booktabs'.~You-should-load~'booktabs',~before-or~
11125     after~'nicematrix'.~Your-key-will-be-ignored.
11126 }
11127 \@@_msg_new:nn { enumitem~not~loaded }
11128 {
11129     enumitem-not-loaded. \\\
11130     You-can't-use-the-command~ \token_to_str:N \tabularnote \
11131     ~because-you-haven't~loaded~'enumitem'.~We-should-load-it~
11132     (before-or-after~'nicematrix').~All-the-commands~
11133     \token_to_str:N \tabularnote \ will-be-ignored-in-the-document.
11134 }

```

```

11135 \@@_msg_new:nn { tikz-without-tikz }
11136 {
11137     TikZ~not~loaded. \\
11138     You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~loaded.~You~
11139     should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.~
11140     If~you~go~on,~your~key~will~be~ignored.
11141 }
11142 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
11143 {
11144     TikZ~not~loaded. \\
11145     You~have~used~the~key~'tikz'~in~the~definition~of~a~
11146     customized~line~(with~'custom-line')~but~TikZ~is~not~loaded.~
11147     You~can~go~on~but~you~will~have~another~error~if~you~actually~
11148     use~that~custom~line.~You~should~load~TikZ~with~
11149     \token_to_str:N \usepackage \{tikz\}.
11150 }
11151 \@@_msg_new:nn { tikz-in-borders-without-tikz }
11152 {
11153     TikZ~not~loaded. \\
11154     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
11155     command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
11156     Your~key~will~be~ignored.~You~should~load~TikZ~with~
11157     \token_to_str:N \usepackage \{tikz\}
11158 }
11159 \@@_msg_new:nn { color-in-custom-line-with-tikz }
11160 {
11161     Erroneous~use.\\
11162     In~a~'custom-line',~you~have~used~both~'tikz'~(or~'dashed')~and~'color',~
11163     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz'~itself).~
11164     The~key~'color'~will~be~ignored.
11165 }
11166 \@@_msg_new:nn { Wrong~last~row }
11167 {
11168     Wrong~number.\\
11169     You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
11170     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
11171     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
11172     last~row~but~you~should~correct~your~code.~You~can~avoid~this~
11173     problem~by~using~'last-row'~without~value~(more~compilations~
11174     might~be~necessary).
11175 }
11176 \@@_msg_new:nn { Yet~in~env }
11177 {
11178     Nested~environments.\\
11179     Environments~of~nicematrix~can't~be~nested.~However~you~
11180     can~insert,~for~example,~a~\{tabular\}~in~a~\{NiceTabular\}~
11181     or~a~\{NiceTabular\}~in~a~\{tabular\}.~You~can~also~compose~
11182     an~environment~of~nicematrix~in~a~box~of~LaTeX~and~insert~
11183     that~box~in~another~environment~of~nicematrix.
11184 }
11185 \@@_msg_new:nn { Outside~math~mode }
11186 {
11187     Outside~math~mode.\\
11188     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
11189     (and~not~in~ \token_to_str:N \vcenter ).
11190 }
11191 \@@_msg_new:nn { One~letter~allowed }
11192 {
11193     Bad~name.\\
11194     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
11195     you~have~used~' \l_keys_value_tl '~Your~key~will~be~ignored.

```

```

11196 }
11197 \@@_msg_new:nn { TabularNote~in~CodeAfter }
11198 {
11199   Environment~\{TabularNote\}~forbidden.\\
11200   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
11201   but~*before*~the~ \token_to_str:N \CodeAfter .~Your~environment~
11202   \{TabularNote\}~will~be~ignored.
11203 }
11204 \@@_msg_new:nn { varwidth~not~loaded }
11205 {
11206   varwidth~not~loaded.\\
11207   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
11208   loaded.~You~should~load~'varwidth',~before~or~after~'nicematrix'.~
11209   Your~column~will~behave~like~'p'.
11210 }
11211 \@@_msg_new:nn { varwidth~not~loaded~in~X }
11212 {
11213   varwidth~not~loaded.\\
11214   You~can't~use~the~key~'V'~in~your~column~'X'~
11215   because~'varwidth'~is~not~loaded.~You~should~load~'varwidth',~
11216   before~or~after~'nicematrix'.~ Your~key~will~be~ignored.
11217 }
11218 \@@_msg_new:nnn { Unknown~key~for~a~rule }
11219 {
11220   Unknown~key.\\
11221   Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
11222   \c_@@_available_keys_str
11223 }
11224 {
11225   The~available~keys~are:~color,~multiplicity,~sep~color,~tikz~
11226   and~total~width~(the~latter~is~meaningful~only~in~cunjunction~with~'tikz').
11227 }

```

If fact, there is also the key dotted but it won't be very useful since we provide \hdottedline, \cdottedline and the letter :.

```

11228 \@@_msg_new:nnn { Unknown~key~for~Block }
11229 {
11230   Unknown~key. \\
11231   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11232   \token_to_str:N \Block .~Your~key~will~be~ignored. \\
11233   \c_@@_available_keys_str
11234 }
11235 {
11236   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
11237   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~name,~
11238   opacity,~rounded~corners,~r,~respect~arraystretch,~rules/width,~t,~T,~tikz,~
11239   transparent~and~vlines.
11240 }
11241 \@@_msg_new:nnn { Unknown~key~for~Brace }
11242 {
11243   Unknown~key.\\
11244   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
11245   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace .~
11246   Your~key~will~be~ignored. \\
11247   \c_@@_available_keys_str
11248 }
11249 {
11250   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
11251   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
11252   right~shorten)~and~yshift.
11253 }

```

```

11254 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
11255 {
11256   Unknown~key.\\
11257   The~key~' \l_keys_key_str '~is~unknown.~It~will~be~ignored. \\
11258   \c_@@_available_keys_str
11259 }
11260 {
11261   The~available~keys~are~(in~alphabetic~order):~
11262   delimiters/color,~
11263   rules~(with~the~subkeys~'color'~and~'width'),~
11264   sub-matrix~(several~subkeys)~
11265   and~xdots~(several~subkeys).~
11266   The~latter~is~for~the~command~ \token_to_str:N \line .
11267 }
11268 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
11269 {
11270   Unknown~key.\\
11271   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11272   \c_@@_available_keys_str
11273 }
11274 {
11275   The~available~keys~are~(in~alphabetic~order):~
11276   create-cell-nodes,~
11277   delimiters/color~and~
11278   sub-matrix~(several~subkeys).
11279 }
11280 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
11281 {
11282   Unknown~key.\\
11283   The~key~' \l_keys_key_str '~is~unknown.~Your~key~will~be~ignored. \\
11284   \c_@@_available_keys_str
11285 }
11286 {
11287   The~available~keys~are~(in~alphabetic~order):~
11288   'delimiters/color',~
11289   'extra-height',~
11290   'hlines',~
11291   'hvlines',~
11292   'left-xshift',~
11293   'name',~
11294   'right-xshift',~
11295   'rules'~(with~the~subkeys~'color'~and~'width'),~
11296   'slim',~
11297   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~and~'right-xshift').
11298 }
11299 \@@_msg_new:nnn { Unknown~key~for~notes }
11300 {
11301   Unknown~key.\\
11302   The~key~' \l_keys_key_str '~is~unknown.~ Your~key~will~be~ignored. \\
11303   \c_@@_available_keys_str
11304 }
11305 {
11306   The~available~keys~are~(in~alphabetic~order):~
11307   bottomrule,~
11308   code-after,~
11309   code-before(+),~
11310   detect-duplicates,~
11311   enumitem-keys,~
11312   enumitem-keys-para,~
11313   para,~
11314   label-in-list,~
11315   label-in-tabular~and~
11316   style.

```

```

11317 }
11318 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
11319 {
11320   Unknown~key.\\
11321   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11322   \token_to_str:N \RowStyle .~Your~key~will~be~ignored. \\
11323   \c_@@_available_keys_str
11324 }
11325 {
11326   The~available~keys~are~(in~alphabetic~order):~
11327   bold,~
11328   cell-space-top-limit(+),~
11329   cell-space-bottom-limit(+),~
11330   cell-space-limits(+),~
11331   color,~
11332   fill~(alias:~rowcolor),~
11333   nb-rows,~
11334   opacity~and~
11335   rounded-corners.
11336 }
11337 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
11338 {
11339   Unknown~key.\\
11340   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
11341   \token_to_str:N \NiceMatrixOptions .~Your~key~will~be~ignored. \\
11342   \c_@@_available_keys_str
11343 }
11344 {
11345   The~available~keys~are~(in~alphabetic~order):~
11346   &~in~blocks,~
11347   allow-duplicate-names,~
11348   ampersand-in-blocks,~
11349   caption-above,~
11350   cell-space-bottom-limit(+),~
11351   cell-space-limits(+),~
11352   cell-space-top-limit(+),~
11353   code-for-first-col(+),~
11354   code-for-first-row(+),~
11355   code-for-last-col(+),~
11356   code-for-last-row(+),~
11357   corners,~
11358   custom-key,~
11359   create-extra-nodes,~
11360   create-medium-nodes,~
11361   create-large-nodes,~
11362   custom-line,~
11363   delimiters~(several~subkeys),~
11364   end-of-row,~
11365   first-col,~
11366   first-row,~
11367   h(v)lines,~
11368   h(v)lines-except-borders,~
11369   last-col,~
11370   last-row,~
11371   left-margin,~
11372   light-syntax,~
11373   light-syntax-expanded,~
11374   matrix/columns-type,~
11375   no-cell-nodes,~
11376   notes~(several~subkeys),~
11377   nullify-dots,~
11378   pgf-node-code,~
11379   renew-dots,~

```

```

11380   renew-matrix,~
11381   respect-arraystretch,~
11382   rounded-corners,~
11383   right-margin,~
11384   rules~(with~the~subkeys~'color'~and~'width'),~
11385   small,~
11386   sub-matrix~(several~subkeys),~
11387   vl原因,~
11388   xdots~(several~subkeys).
11389 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

11390 \@@_msg_new:nnn { Unknown-key-for~NiceArray }
11391 {
11392   Unknown-key.\\
11393   The-key~' \l_keys_key_str '~is-unknown-for-the-environment~
11394   \{NiceArray\}.~Your-key-will-be-ignored. \\
11395   \c_@@_available_keys_str
11396 }
11397 {
11398   The-available-keys-are~(in~alphabetic-order):~
11399   &-in-blocks,~
11400   ampersand-in-blocks,~
11401   b,~
11402   baseline,~
11403   c,~
11404   cell-space-bottom-limit,~
11405   cell-space-limits,~
11406   cell-space-top-limit,~
11407   code-after,~
11408   code-for-first-col(+),~
11409   code-for-first-row(+),~
11410   code-for-last-col(+),~
11411   code-for-last-row(+),~
11412   columns-width,~
11413   corners,~
11414   create-blocks-in-col,~
11415   create-extra-nodes,~
11416   create-medium-nodes,~
11417   create-large-nodes,~
11418   draw-trees-in-col,~
11419   extra-left-margin,~
11420   extra-right-margin,~
11421   first-col,~
11422   first-row,~
11423   h(v)lines,~
11424   h(v)lines-except-borders,~
11425   last-col,~
11426   last-row,~
11427   left-margin,~
11428   light-syntax,~
11429   light-syntax-expanded,~
11430   name,~
11431   no-cell-nodes,~
11432   nullify-dots,~
11433   pgf-node-code,~
11434   renew-dots,~
11435   respect-arraystretch,~
11436   right-margin,~
11437   rounded-corners,~
11438   rules~(with~the~subkeys~'color'~and~'width'),~
11439   small,~
11440   t,~

```



```

11441     vlines,~
11442     xdots/color,~
11443     xdots/shorten-start(+),~
11444     xdots/shorten-end(+),~
11445     xdots/shorten(+)-and~
11446     xdots/line-style.
11447 }

```

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`).

```

11448 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
11449 {
11450   Unknown~key.\\
11451   The~key~' \l_keys_key_str 'is~unknown~for~the~
11452   \@@_full_name_env: .~Your~key~will~be~ignored. \\
11453   \c_@@_available_keys_str
11454 }
11455 {
11456   The~available~keys~are~(in~alphabetic~order):~
11457   &~in~blocks,~
11458   ampersand~in~blocks,~
11459   b,~
11460   baseline,~
11461   c,~
11462   cell-space-bottom-limit,~
11463   cell-space-limits,~
11464   cell-space-top-limit,~
11465   code-after,~
11466   code-for-first-col(+),~
11467   code-for-first-row(+),~
11468   code-for-last-col(+),~
11469   code-for-last-row(+),~
11470   columns-type,~
11471   columns-width,~
11472   corners,~
11473   create-blocks-in-col,~
11474   create-extra-nodes,~
11475   create-medium-nodes,~
11476   create-large-nodes,~
11477   draw-trees-in-col,~
11478   extra-left-margin,~
11479   extra-right-margin,~
11480   first-col,~
11481   first-row,~
11482   h(v)lines,~
11483   h(v)lines-except-borders,~
11484   l,~
11485   last-col,~
11486   last-row,~
11487   left-margin,~
11488   light-syntax,~
11489   light-syntax-expanded,~
11490   name,~
11491   no-cell-nodes,~
11492   nullify-dots,~
11493   pgf-node-code,~
11494   r,~
11495   renew-dots,~
11496   respect-arraystretch,~
11497   right-margin,~
11498   rounded-corners,~
11499   rules~(with~the~subkeys~'color'~and~'width'),~
11500   small,~

```

```

11501     t,~
11502     vlines,~
11503     xdots/color,~
11504     xdots/shorten-start(+),~
11505     xdots/shorten-end(+),~
11506     xdots/shorten(+)-and~
11507     xdots/line-style.
11508 }

11509 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
11510 {
11511     Unknown~key.\\
11512     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
11513     \{NiceTabular\}.~Your~key~will~be~ignored. \\
11514     \c_@@_available_keys_str
11515 }
11516 {
11517     The~available~keys~are~(in~alphabetic~order):~
11518     &in~blocks,~
11519     ampersand~in~blocks,~
11520     b,~
11521     baseline,~
11522     c,~
11523     caption,~
11524     cell-space-bottom-limit,~
11525     cell-space-limits,~
11526     cell-space-top-limit,~
11527     code-after,~
11528     code-for-first-col(+),~
11529     code-for-first-row(+),~
11530     code-for-last-col(+),~
11531     code-for-last-row(+),~
11532     columns-width,~
11533     corners,~
11534     custom-line,~
11535     create-blocks-in-col,~
11536     create-extra-nodes,~
11537     create-medium-nodes,~
11538     create-large-nodes,~
11539     draw-trees-in-col,~
11540     extra-left-margin,~
11541     extra-right-margin,~
11542     first-col,~
11543     first-row,~
11544     h(v)lines,~
11545     h(v)lines-except-borders,~
11546     label,~
11547     last-col,~
11548     last-row,~
11549     left-margin,~
11550     light-syntax,~
11551     light-syntax-expanded,~
11552     name,~
11553     no-cell-nodes,~
11554     notes~(several~subkeys),~
11555     nullify-dots,~
11556     pgf-node-code,~
11557     renew-dots,~
11558     respect-arraystretch,~
11559     right-margin,~
11560     rounded-corners,~
11561     rules~(with~the~subkeys~'color'~and~'width'),~
11562     short-caption,~
11563     t,~

```

```

11564     tabularnote,~
11565     vlines,~
11566     xdots/color,~
11567     xdots/shorten-start(+),~
11568     xdots/shorten-end(+),~
11569     xdots/shorten(+)-and~
11570     xdots/line-style.
11571 }

11572 \@@_msg_new:nnn { Duplicate~name }
11573 {
11574     Duplicate~name.\\
11575     The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
11576     the~same~environment~name~twice.~You~can~go~on,~but,~
11577     maybe,~you~will~have~incorrect~results~especially~
11578     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11579     message~again,~use~the~key~'allow-duplicate-names'~in~
11580     ' \token_to_str:N \NiceMatrixOptions '.\\
11581     \bool_if:NF \g_@@_messages_for_Overleaf_bool
11582     { For~a~list~of~the~names~already~used,~type~H~<return>. }
11583 }
11584 {
11585     The~names~already~defined~in~this~document~are:~
11586     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11587 }

11588 \@@_msg_new:nn { caption-above~in~env }
11589 {
11590     The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.~
11591     Your~key~will~be~ignored.
11592 }

11593 \@@_msg_new:nn { show-cell-names }
11594 {
11595     There~is~no~key~'show-cell-names'~in~nicematrix.\\
11596     You~should~use~the~command~\token_to_str:N \ShowCellNames\
11597     in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11598     \CodeAfter.~Your~key~will~be~ignored.
11599 }

11600 \@@_msg_new:nn { Option~auto~for~columns-width }
11601 {
11602     Erroneous~use.\\
11603     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11604     Your~key~will~be~ignored.
11605 }

11606 \@@_msg_new:nn { NiceTabularX~without~X }
11607 {
11608     NiceTabularX~without~X.\\
11609     You~should~not~use~\{NiceTabularX\}~without~X~columns.~However,~you~can~go~on.
11610 }

11611 \@@_msg_new:nn { Preamble~forgotten }
11612 {
11613     Preamble~forgotten.\\
11614     You~have~probably~forgotten~the~preamble~of~your~
11615     \@@_full_name_env: .
11616 }

11617 \@@_msg_new:nn { Invalid~col~number }
11618 {
11619     Invalid~column~number.\\
11620     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11621     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.~
11622     Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11623 }

```

```

11624 \@@_msg_new:nn { Invalid~row~number }
11625 {
11626   Invalid~row~number.\\
11627   A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11628   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.~
11629   Maybe~this~is~a~spurious~error~due~to~an~incorrect~'aux'~file.
11630 }

11631 \@@_define_com:NNN p ( )
11632 \@@_define_com:NNN b [ ]
11633 \@@_define_com:NNN v | |
11634 \@@_define_com:NNN V \l \l
11635 \@@_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	183
27	Automatic arrays	211
28	The redefinition of the command <code>\dotfill</code>	212
29	The command <code>\diagbox</code>	212

30	The keyword <code>\CodeAfter</code>	214
31	The delimiters in the preamble	214
32	The command <code>\SubMatrix</code>	216
33	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	225
34	The commands <code>HBrace</code> et <code>VBrace</code>	228
35	The command <code>TikzEveryCell</code>	231
36	The key <code>draw-trees-in-col</code>	232
37	The key <code>create-blocks-in-col</code>	234
38	The command <code>\ShowCellNames</code>	234
39	We process the options at package loading	236
40	About the package underscore	238
41	Compatibility with <code>threeparttable</code>	238
42	Error messages of the package	238