

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

June 21, 2025

## 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 traduction: `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-a-the-version-of-2023-11-01
12 }

13 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
14 \IfFormatAtLeastTF
15 { 2023-11-01 }
16 { }
17 { \msg_fatal:nn { nicematrix } { latex-too-old } }

18 \ProvideDocumentCommand { \IfPackageLoadedT } { m m }
19 { \IfPackageLoadedTF { #1 } { #2 } { } }
20
21 \ProvideDocumentCommand { \IfPackageLoadedF } { m m }
22 { \IfPackageLoadedTF { #1 } { } { #2 } }
```

---

\*This document corresponds to the version 7.1d of `nicematrix`, at the date of 2025/06/21.

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

```
23 \RequirePackage { amsmath }
```

```
24 \RequirePackage { array }
```

In the version 2.6a of array, important modifications have been done for the Tagging Project.

```
25 \bool_const:Nn \c_@@_recent_array_bool
26 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
27 \bool_const:Nn \c_@@_testphase_table_bool
28 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

```
29 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
30 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
31 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
32 \cs_generate_variant:Nn \@@_error:nn { n e }
33 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
35 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
36 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

With Overleaf, 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).

```
37 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
38 {
39   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
40     { \msg_new:nnn { nicematrix } { #1 } { #2 \\\ #3 } }
41     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
42 }
```

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

```
43 \cs_new_protected:Npn \@@_error_or_warning:n
44 {
45   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
46     { \@@_warning:n }
47     { \@@_error:n }
48 }
```

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”.

```
49 \bool_new:N \g_@@_messages_for_Overleaf_bool
50 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
51 {
52   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
53   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
54 }
```

```
55 \@@_msg_new:nn { mdwtab-loaded }
56 {
57   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
58   This~error~is~fatal.
59 }
```

```
60 \hook_gput_code:nnn { begindocument / end } { . }
61 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab-loaded } } }
```

## 2 Collecting options

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

*Example :*

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

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

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,

the command `\G` takes in an arbitrary number of optional arguments between square brackets.

Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

62 \cs_new_protected:Npn \@@_collect_options:n #1
63 {
64   \peek_meaning:NTF [
65     { \@@_collect_options:nw { #1 } }
66     { #1 { } }
67   }

```

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

```

68 \NewDocumentCommand \@@_collect_options:nw { m r[] }
69 { \@@_collect_options:nn { #1 } { #2 } }
70
71 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
72 {
73   \peek_meaning:NTF [
74     { \@@_collect_options:nnw { #1 } { #2 } }
75     { #1 { #2 } }
76   }
77
78 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
79 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

## 3 Technical definitions

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

```

80 \tl_const:Nn \c_@@_b_tl { b }
81 \tl_const:Nn \c_@@_c_tl { c }
82 \tl_const:Nn \c_@@_l_tl { l }
83 \tl_const:Nn \c_@@_r_tl { r }
84 \tl_const:Nn \c_@@_all_tl { all }
85 \tl_const:Nn \c_@@_dot_tl { . }
86 \str_const:Nn \c_@@_r_str { r }
87 \str_const:Nn \c_@@_c_str { c }
88 \str_const:Nn \c_@@_l_str { l }

```

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).

```

89 \tl_new:N \l_@@_argspec_tl

```

```

90 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
91 \cs_generate_variant:Nn \str_set:Nn { N o }
92 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
93 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
94 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
95 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
96 \cs_generate_variant:Nn \dim_min:nn { v }
97 \cs_generate_variant:Nn \dim_max:nn { v }

98 \hook_gput_code:nnn { begindocument } { . }
99 {
100   \IfPackageLoadedTF { tikz }
101   {

```

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 `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

102     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
103     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
104   }
105   {
106     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
107     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
108   }
109 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2025, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

110 \IfClassLoadedTF { revtex4-1 }
111 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112 {
113   \IfClassLoadedTF { revtex4-2 }
114   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
115   {

```

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

```

116     \cs_if_exist:NT \rvtx@ifformat@geq
117     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
118     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
119   }
120 }

```

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.

```

121 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
122 {
123   \iow_now:Nn \@mainaux
124   {
125     \ExplSyntaxOn
126     \cs_if_free:NT \pgfsyspdfmark
127     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
128     \ExplSyntaxOff
129   }
130   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
131 }

```

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.

```

132 \ProvideDocumentCommand \iddots { }
133 {
134   \mathinner
135   {
136     \mkern 1 mu
137     \box_move_up:nn { 1 pt } { \hbox { . } }
138     \mkern 2 mu
139     \box_move_up:nn { 4 pt } { \hbox { . } }
140     \mkern 2 mu
141     \box_move_up:nn { 7 pt }
142     { \vbox:n { \kern 7 pt \hbox { . } } }
143     \mkern 1 mu
144   }
145 }

```

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.

```

146 \hook_gput_code:nnn { begindocument } { . }
147 {
148   \IfPackageLoadedT { booktabs }
149   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
150 }
151 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
152 {
153   \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`).

```

154   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
155   {

```

`\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

156     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
157     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
158   }
159 }

```

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`.

```

160 \hook_gput_code:nnn { begindocument } { . }
161 {
162   \cs_set_protected:Npe @@_everycr:
163   {
164     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
165     { \noalign { \@@_in_everycr: } }
166   }
167   \IfPackageLoadedTF { colortbl }
168   {
169     \cs_set_eq:NN @@_old_cellcolor: \cellcolor
170     \cs_set_eq:NN @@_old_rowcolor: \rowcolor
171     \cs_new_protected:Npn @@_revert_colortbl:
172     {
173       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
174       {
175         \cs_set_eq:NN \cellcolor @@_old_cellcolor:
176         \cs_set_eq:NN \rowcolor @@_old_rowcolor:

```

```

177     }
178 }

```

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).

```

179 \cs_new_protected:Npn \@@_replace_columncolor:
180 {
181     \tl_replace_all:Nnn \g_@@_array_preamble_tl
182     { \columncolor }
183     { \@@_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.

```

184     }
185 }
186 {
187     \cs_new_protected:Npn \@@_revert_colortbl: { }
188     \cs_new_protected:Npn \@@_replace_columncolor:
189     { \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.

```

190 \def \CT@arc@ { }
191 \def \arrayrulecolor #1 # { \CT@arc { #1 } }
192 \def \CT@arc #1 #2
193 {
194     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
195     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
196 }

```

Idem for `\CT@drs@`.

```

197 \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
198 \def \CT@drs #1 #2
199 {
200     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
201     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
202 }
203 \def \hline
204 {
205     \noalign { \ifnum 0 = ` } \fi
206     \cs_set_eq:NN \hskip \vskip
207     \cs_set_eq:NN \vrule \hrule
208     \cs_set_eq:NN \@width \@height
209     { \CT@arc@ \vline }
210     \futurelet \reserved@a
211     \@xhline
212 }
213 }
214 }

```

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.

```

215 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
216 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
217 {
218     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
219     \int_compare:nNnT { #1 } > { \c_one_int }
220     { \multispan { \int_eval:n { #1 - 1 } } & }
221     \multispan { \int_eval:n { #2 - #1 + 1 } }
222     {
223         \CT@arc@
224         \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>

```
225     \skip_horizontal:N \c_zero_dim
226 }
```

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`.

```
227     \everycr { }
228     \cr
229     \noalign { \skip_vertical:n { - \arrayrulewidth } }
230 }
```

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.

```
231 \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`.

```
232 { \@@_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*.

```
233 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
234 \cs_generate_variant:Nn \@@_cline_i:nn { e }
235 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
236 {
237     \tl_if_empty:nTF { #3 }
238     { \@@_cline_iii:w #1|#2-#2 \q_stop }
239     { \@@_cline_ii:w #1|#2-#3 \q_stop }
240 }
241 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
242 { \@@_cline_iii:w #1|#2-#3 \q_stop }
243 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
244 {
```

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).

```
245     \int_compare:nNnT { #1 } < { #2 }
246     { \multispan { \int_eval:n { #2 - #1 } } & }
247     \multispan { \int_eval:n { #3 - #2 + 1 } }
248     {
249         \CT@arc@
250         \leaders \hrule \@height \arrayrulewidth \hfill
251         \skip_horizontal:N \c_zero_dim
252     }
```

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

```
253     \peek_meaning_remove_ignore_spaces:NNTF \cline
254     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
255     { \everycr { } \cr }
256 }
```

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

```
257 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

---

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

```

258 \cs_new_protected:Npn \@@_set_CTarc:n #1
259 {
260   \tl_if_blank:nF { #1 }
261   {
262     \tl_if_head_eq_meaning:nNTF { #1 } [
263       { \def \CT@arc@ { \color #1 } }
264       { \def \CT@arc@ { \color { #1 } } }
265     ]
266   }
267 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

268 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
269 {
270   \tl_if_head_eq_meaning:nNTF { #1 } [
271     { \def \CT@drsc@ { \color #1 } }
272     { \def \CT@drsc@ { \color { #1 } } }
273   ]
274 \cs_generate_variant:Nn \@@_set_CTdrsc:n { o }

```

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

```

275 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
276 {
277   \tl_if_head_eq_meaning:nNTF { #2 } [
278     { #1 #2 }
279     { #1 { #2 } }
280   ]
281 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

282 \cs_new_protected:Npn \@@_color:n #1
283 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
284 \cs_generate_variant:Nn \@@_color:n { o }

```

```

285 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
286 {
287   \tl_set_rescan:Nno
288     #1
289     {
290       \char_set_catcode_other:N >
291       \char_set_catcode_other:N <
292     }
293   #1
294 }

```

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

```

295 \dim_new:N \l_@@_tmpc_dim
296 \dim_new:N \l_@@_tmpd_dim
297 \dim_new:N \l_@@_tmpe_dim
298 \dim_new:N \l_@@_tmpf_dim
299 \tl_new:N \l_@@_tmpc_tl
300 \tl_new:N \l_@@_tmpd_tl
301 \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.

```
302 \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).

```
303 \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.

```
304 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
305 { \int_use:N \g_@@_env_int }
```

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

```
306 \cs_new_protected:Npn \@@_qpoint:n #1
307 { \pgfpointanchor { \@@_env: - #1 } { center } }
```

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

```
308 \bool_new:N \l_@@_tabular_bool
```

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

```
309 \bool_new:N \g_@@_delims_bool
310 \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.

```
311 \bool_new:N \l_@@_preamble_bool
312 \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.

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

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

```
314 \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.

```
315 \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).

```
316 \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.).

```
317 \dim_new:N \l_@@_col_width_dim
318 \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.

```
319 \int_new:N \g_@@_row_total_int
320 \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).

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

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

```
322 \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.

```
323 \tl_new:N \l_@@_hpos_cell_tl
324 \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`.

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

Idem for the mono-row blocks.

```
326 \dim_new:N \g_@@_blocks_ht_dim
327 \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}`).

```
328 \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`.

```
329 \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.

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

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

```
331 \bool_new:N \l_@@_notes_detect_duplicates_bool
332 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
333 \bool_new:N \l_@@_initial_open_bool
334 \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.

```
335 \dim_new:N \l_@@_tabular_width_dim
```

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```
336 \dim_new:N \l_@@_rule_width_dim
```

The key `color` in a command of rule such as `\Hline` (or the specifier “|” in the preamble of an environment).

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

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

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

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

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

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

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

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

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

```
342 \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 }`).

```
343 \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.

```
344 \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.

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

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

```
347 \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}`).

```
348 \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`).

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

For `\multicolumn`.

```
350 \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`.

```
351 \tl_new:N \l_@@_columns_type_tl
352 \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 `:`.

```
353 \tl_new:N \l_@@_xdots_down_tl
354 \tl_new:N \l_@@_xdots_up_tl
355 \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`).

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

357 \cs_new_protected:Npn \@@_test_if_math_mode:
358 {
359   \if_mode_math: \else:
360     \@@_fatal:n { Outside-math-mode }
361   \fi:
362 }
```

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 analyse of the preamble of the array.

```
363 \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”.

```
364 \colorlet { nicematrix-last-col } { . }
365 \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`).

```
366 \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*.

```
367 \tl_new:N \g_@@_com_or_env_str
368 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

```
370 \cs_new:Npn \@@_full_name_env:
371 {
372   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
373     { command \space \c_backslash_str \g_@@_name_env_str }
374     { environment \space \{ \g_@@_name_env_str \} }
375 }
```

```
376 \tl_new:N \g_@@_cell_after_hook_tl % 2025/03/22
```

For the key `code` of the command `\SubMatrix` (itself in the main `\CodeAfter`), we will use the following token list.

```
377 \tl_new:N \l_@@_code_tl
```

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form `i-j`) will be created.

```
378 \tl_new:N \l_@@_pgf_node_code_tl
```

The so-called `\CodeBefore` is split in two parts because we want to control the order of execution of some instructions.

```
379 \tl_new:N \g_@@_pre_code_before_tl
380 \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`.

The so-called `\CodeAfter` is split in two parts because we want to control the order of execution of some instructions.

```
381 \tl_new:N \g_@@_pre_code_after_tl
382 \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.

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

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

```
384 \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.

```
385 \int_new:N \l_@@_old_iRow_int
386 \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).

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

The following token list corresponds to the key `rules/color` available in the environments.

```
388 \tl_new:N \l_@@_rules_color_tl
```

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

```
389 \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.

```
390 \bool_new:N \l_@@_X_columns_aux_bool
391 \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.

```
392 \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).

```
393 \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).

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

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

```
396 \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).

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

The following dimensions will be used when drawing the dotted lines.

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

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

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

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

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

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

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

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

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

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

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

```
408 \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”.

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

```
410 \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.

```
411 \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!).

```
412 \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.

```
413 \seq_new:N \g_@@_future_pos_of_blocks_seq
```

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocs_seq` will be erased and replaced by the value of `\g_@@_future_pos_of_blocks_seq`.

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}`.

```
414 \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.

```
415 \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`).

```
416 \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`).

```
417 \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.

```
418 \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).

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

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

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.

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

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

---

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

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  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.

```

423 \dim_new:N \g_@@_delta_x_one_dim
424 \dim_new:N \g_@@_delta_y_one_dim
425 \dim_new:N \g_@@_delta_x_two_dim
426 \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`).

```

427 \int_new:N \l_@@_row_min_int
428 \int_new:N \l_@@_row_max_int
429 \int_new:N \l_@@_col_min_int
430 \int_new:N \l_@@_col_max_int

431 \int_new:N \l_@@_initial_i_int
432 \int_new:N \l_@@_initial_j_int
433 \int_new:N \l_@@_final_i_int
434 \int_new:N \l_@@_final_j_int

```

The following counters will be used when drawing the rules.

```

435 \int_new:N \l_@@_start_int
436 \int_set_eq:NN \l_@@_start_int \c_one_int
437 \int_new:N \l_@@_end_int
438 \int_new:N \l_@@_local_start_int
439 \int_new:N \l_@@_local_end_int

```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. Each sub-matrix is represented by an “object” of the form `{i}{j}{k}{l}` where  $i$  and  $j$  are the number of row and column of the upper-left cell and  $k$  and  $l$  the number of row and column of the lower-right cell.

```

440 \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).

```

441 \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`.

```

442 \tl_new:N \l_@@_fill_tl
443 \tl_new:N \l_@@_opacity_tl
444 \tl_new:N \l_@@_draw_tl
445 \seq_new:N \l_@@_tikz_seq
446 \clist_new:N \l_@@_borders_clist
447 \dim_new:N \l_@@_rounded_corners_dim

```

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 following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```

448 \dim_new:N \l_@@_tab_rounded_corners_dim

```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

```

449 \tl_new:N \l_@@_color_tl

```



In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of tikz keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

```
450 \dim_new:N \l_@@_offset_dim
```

Here is the dimension for the width of the rule when a block (created by `\Block`) is stroked or when the key `hvlines` is used.

```
451 \dim_new:N \l_@@_line_width_dim
```

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`).

```
452 \str_new:N \l_@@_hpos_block_str
453 \str_set:Nn \l_@@_hpos_block_str { c }
454 \bool_new:N \l_@@_hpos_of_block_cap_bool
455 \bool_new:N \l_@@_p_block_bool
```

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

```
456 \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).

```
457 \str_new:N \l_@@_vpos_block_str
```

Used when the key `draw-first` is used for `\Ddots` or `\Iddots`.

```
458 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
459 \bool_new:N \l_@@_vlines_block_bool
460 \bool_new:N \l_@@_hlines_block_bool
```

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

```
461 \int_new:N \g_@@_block_box_int

462 \dim_new:N \l_@@_submatrix_extra_height_dim
463 \dim_new:N \l_@@_submatrix_left_xshift_dim
464 \dim_new:N \l_@@_submatrix_right_xshift_dim
465 \clist_new:N \l_@@_hlines_clist
466 \clist_new:N \l_@@_vlines_clist
467 \clist_new:N \l_@@_submatrix_hlines_clist
468 \clist_new:N \l_@@_submatrix_vlines_clist
```

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}`).

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

The following flag will be used by (for instance) `\@@_vline_ii:`. When `\l_@@_dotted_bool` is `true`, a dotted line (with our system) will be drawn.

```
470 \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`).

```
471 \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.

```
472 \int_new:N \l_@@_first_row_int
473 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

- **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.

```
474 \int_new:N \l_@@_first_col_int
475 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

- **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).

```
476 \int_new:N \l_@@_last_row_int
477 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
479 \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.

```
480 \int_new:N \l_@@_last_col_int
481 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

---

<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.

```

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

```

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

```

482 \bool_new:N \g_@@_last_col_found_bool

```

This boolean is set to `false` at the end of `\@@_pre_array_ii:`.

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

```

483 \bool_new:N \l_@@_in_last_col_bool

```

## Some utilities

```

484 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
485 {

```

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

```

486 \def \l_tmpa_tl { #1 }
487 \def \l_tmpb_tl { #2 }
488 }

```

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.

```

489 \cs_new_protected:Npn \@@_expand_clist:N #1
490 {
491 \clist_if_in:NnF #1 { all }
492 {
493 \clist_clear:N \l_tmpa_clist
494 \clist_map_inline:Nn #1
495 {

```

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

```

496 \tl_if_in:nnTF { ##1 } { - }
497 { \@@_cut_on_hyphen:w ##1 \q_stop }
498 {

```

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

```

499 \def \l_tmpa_tl { ##1 }
500 \def \l_tmpb_tl { ##1 }
501 }
502 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
503 { \clist_put_right:Nn \l_tmpa_clist { #####1 } }
504 }
505 \tl_set_eq:NN #1 \l_tmpa_clist
506 }
507 }

```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- `\Vdots` with both extremities open (and hence also `\Vdotsfor` in an exterior column;
- 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.

```

508 \hook_gput_code:nnn { begindocument } { . }
509 {
510 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
511 \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
512 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
513 }

```

## 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 `\c_novalue_tl`).
  - 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.

```
514 \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`.

```
515 \int_new:N \g_@@_tabularnote_int
516 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
517 \seq_new:N \g_@@_notes_seq
518 \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.

```
519 \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.

```
520 \seq_new:N \l_@@_notes_labels_seq
521 \newcounter { nicematrix_draft }
```

---

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

```

522 \cs_new_protected:Npn \@@_notes_format:n #1
523 {
524   \setcounter { nicematrix_draft } { #1 }
525   \@@_notes_style:n { nicematrix_draft }
526 }

```

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

```

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

```

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

```

528 \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`.

```

529 \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`.

```

530 \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`).

```

531 \hook_gput_code:nnn { begindocument } { . }
532 {
533   \IfPackageLoadedTF { enumitem }
534   {

```

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.

```

535     \newlist { tabularnotes } { enumerate } { 1 }
536     \setlist [ tabularnotes ]
537     {
538       topsep = 0pt ,
539       noitemsep ,
540       leftmargin = * ,
541       align = left ,
542       labelsep = 0pt ,
543       label =
544         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
545     }
546     \newlist { tabularnotes* } { enumerate* } { 1 }
547     \setlist [ tabularnotes* ]
548     {
549       afterlabel = \nobreak ,
550       itemjoin = \quad ,
551       label =
552         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
553     }

```

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`.

```

554     \NewDocumentCommand \tabularnote { o m }
555     {
556       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } { \l_@@_in_env_bool }

```

```

557         {
558             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
559             { \@@_error:n { tabularnote~forbidden } }
560             {
561                 \bool_if:NTF \l_@@_in_caption_bool
562                 \@@_tabularnote_caption:nn
563                 \@@_tabularnote:nn
564                 { #1 } { #2 }
565             }
566         }
567     }
568 }
569 {
570     \NewDocumentCommand \tabularnote { o m }
571     { \@@_err_enumitem_not_loaded: }
572 }
573 }
574 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
575 {
576     \@@_error_or_warning:n { enumitem~not~loaded }
577     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
578 }
579 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
580 { \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 `\c_novalue_tl`) and #2 is the mandatory argument of `\tabularnote`.

```

581 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
582 {

```

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).

```

583     \int_zero:N \l_tmpa_int
584     \bool_if:NNT \l_@@_notes_detect_duplicates_bool
585     {

```

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

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

If the user have used `\tabularnote` without the optional argument, the `label` will be the special marker expressed by `\c_novalue_tl`.

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabularnote`.

```

586     \int_zero:N \l_tmpb_int
587     \seq_map_indexed_inline:Nn \g_@@_notes_seq
588     {
589         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
590         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
591         {
592             \tl_if_novalue:nTF { #1 }
593             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
594             { \int_set:Nn \l_tmpa_int { ##1 } }
595             \seq_map_break:
596         }
597     }
598     \int_if_zero:nF { \l_tmpa_int }
599     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }

```

```

600     }
601     \int_if_zero:nT { \l_tmpa_int }
602     {
603         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
604         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
605     }
606     \seq_put_right:Ne \l_@@_notes_labels_seq
607     {
608         \tl_if_novalue:nTF { #1 }
609         {
610             \@@_notes_format:n
611             {
612                 \int_eval:n
613                 {
614                     \int_if_zero:nTF { \l_tmpa_int }
615                     { \c@tabularnote }
616                     { \l_tmpa_int }
617                 }
618             }
619         }
620         { #1 }
621     }
622     \peek_meaning:NF \tabularnote
623     {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```

624     \hbox_set:Nn \l_tmpa_box
625     {

```

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

```

626         \@@_notes_label_in_tabular:n
627         {
628             \seq_use:Nnnn
629             \l_@@_notes_labels_seq { , } { , } { , }
630         }
631     }

```

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

```

632     \int_gdecr:N \c@tabularnote
633     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

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

```

634     \int_gincr:N \g_@@_tabularnote_int
635     \refstepcounter { tabularnote }
636     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
637     { \int_gincr:N \c@tabularnote }
638     \seq_clear:N \l_@@_notes_labels_seq
639     \bool_lazy_or:nnTF
640     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
641     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
642     {
643         \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).

```

644         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
645     }
646     { \box_use:N \l_tmpa_box }
647 }

```

```
648 }
```

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`.

```
649 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
650 {
651   \bool_if:NTF \g_@@_caption_finished_bool
652   {
653     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
654     { \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`!

```
655     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
656     { \@@_error:n { Identical~notes~in~caption } }
657   }
658 }
```

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

```
659     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
660     {
```

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.

```
661       \bool_gset_true:N \g_@@_caption_finished_bool
662       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
663       \int_gzero:N \c@tabularnote
664     }
665     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
666 }
```

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!

```
667   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
668   \seq_put_right:Ne \l_@@_notes_labels_seq
669   {
670     \tl_if_novalue:nTF { #1 }
671     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
672     { #1 }
673   }
674   \peek_meaning:NF \tabularnote
675   {
676     \@@_notes_label_in_tabular:n
677     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
678     \seq_clear:N \l_@@_notes_labels_seq
679   }
680 }

681 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
682 { \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.

```

683 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
684 {
685   \begin { pgfscope }
686   \pgfset
687   {
688     inner~sep = \c_zero_dim ,
689     minimum~size = \c_zero_dim
690   }
691   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
692   \pgfnode
693   { rectangle }
694   { center }
695   {
696     \vbox_to_ht:nn
697     { \dim_abs:n { #5 - #3 } }
698     {
699       \vfill
700       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
701     }
702   }
703   { #1 }
704   { }
705   \end { pgfscope }
706 }

```

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.

```

707 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
708 {
709   \begin { pgfscope }
710   \pgfset
711   {
712     inner~sep = \c_zero_dim ,
713     minimum~size = \c_zero_dim
714   }
715   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
716   \pgfpointdiff { #3 } { #2 }
717   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
718   \pgfnode
719   { rectangle }
720   { center }
721   {
722     \vbox_to_ht:nn
723     { \dim_abs:n \l_tmpb_dim }
724     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
725   }
726   { #1 }
727   { }
728   \end { pgfscope }
729 }

```

## 7 The options

The following parameter corresponds to the keys `caption`, `short-caption` and `label` of the environment `{NiceTabular}`.

```

730 \tl_new:N \l_@@_caption_tl

```

```

731 \tl_new:N \l_@@_short_caption_tl
732 \tl_new:N \l_@@_label_tl

```

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).

```

733 \bool_new:N \l_@@_caption_above_bool

```

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 `\l_@@_standard_line_bool`.

```

734 \bool_new:N \l_@@_standard_cline_bool

```

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

```

735 \dim_new:N \l_@@_cell_space_top_limit_dim
736 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

The following parameter corresponds to the key `xdots/horizontal_labels`.

```

737 \bool_new:N \l_@@_xdots_h_labels_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.

```

738 \dim_new:N \l_@@_xdots_inter_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \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).

```

741 \dim_new:N \l_@@_xdots_shorten_start_dim
742 \dim_new:N \l_@@_xdots_shorten_end_dim
743 \hook_gput_code:nnn { begindocument } { . }
744 {
745   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
746   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
747 }

```

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.

```

748 \dim_new:N \l_@@_xdots_radius_dim
749 \hook_gput_code:nnn { begindocument } { . }
750 { \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.

```

751 \tl_new:N \l_@@_xdots_line_style_tl
752 \tl_const:Nn \c_@@_standard_tl { standard }
753 \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`.

```
754 \bool_new:N \l_@@_light_syntax_bool
755 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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).

```
756 \tl_new:N \l_@@_baseline_tl
757 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```
758 \bool_new:N \l_@@_amp_in_blocks_bool
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

```
759 \bool_new:N \l_@@_exterior_arraycolsep_bool
```

The flag `\l_@@_parallelize_diags_bool` controls whether the diagonals are parallelized. The initial value is `true`.

```
760 \bool_new:N \l_@@_parallelize_diags_bool
761 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be within NW, SW, NE and SE.

```
762 \clist_new:N \l_@@_corners_clist
```

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, 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.).

```
763 \bool_new:N \l_@@_nullify_dots_bool
```

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

```
764 \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).

```
765 \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!

```
766 \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.

```
767 \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”.

```
768 \bool_new:N \l_@@_medium_nodes_bool
769 \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).

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

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
771 \dim_new:N \l_@@_left_margin_dim
772 \dim_new:N \l_@@_right_margin_dim
```

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

```
773 \dim_new:N \l_@@_extra_left_margin_dim
774 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
775 \tl_new:N \l_@@_end_of_row_tl
776 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

```
777 \tl_new:N \l_@@_xdots_color_tl
```

The following token list corresponds to the key `delimiters/color`.

```
778 \tl_new:N \l_@@_delimiters_color_tl
```

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. The following boolean corresponds to this option.

```
779 \bool_new:N \l_@@_delimiters_max_width_bool
```

```
780 \keys_define:nn { nicematrix / xdots }
781 {
782   shorten-start .code:n =
783     \hook_gput_code:nnn { begindocument } { . }
784     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
785   shorten-end .code:n =
786     \hook_gput_code:nnn { begindocument } { . }
787     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
788   shorten-start .value_required:n = true ,
789   shorten-end .value_required:n = true ,
790   shorten .code:n =
791     \hook_gput_code:nnn { begindocument } { . }
792     {
793       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
794       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
795     } ,
796   shorten .value_required:n = true ,
797   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
798   horizontal-labels .default:n = true ,
799   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
800   horizontal-label .default:n = true ,
801   line-style .code:n =
802   {
```

```

803     \bool_lazy_or:nnTF
804     { \cs_if_exist_p:N \tikzpicture }
805     { \str_if_eq_p:nn { #1 } { standard } }
806     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
807     { \@@_error:n { bad-option-for-line-style } }
808   } ,
809   line-style .value_required:n = true ,
810   color .tl_set:N = \l_@@_xdots_color_tl ,
811   color .value_required:n = true ,
812   radius .code:n =
813     \hook_gput_code:nnn { begindocument } { . }
814     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
815   radius .value_required:n = true ,
816   inter .code:n =
817     \hook_gput_code:nnn { begindocument } { . }
818     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
819   radius .value_required:n = true ,

```

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

```

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

```

823   draw-first .code:n = \prg_do_nothing: ,
824   unknown .code:n = \@@_error:n { Unknown-key-for-~xdots }
825 }

```

```

826 \keys_define:nn { nicematrix / rules }
827 {
828   color .tl_set:N = \l_@@_rules_color_tl ,
829   color .value_required:n = true ,
830   width .dim_set:N = \arrayrulewidth ,
831   width .value_required:n = true ,
832   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
833 }
834 \cs_new_protected:Npn \@@_err_key_color_inside:
835 {
836   \@@_warning:n { key~color-inside }
837   \cs_gset:Npn \@@_err_key_color_inside: { }
838 }

```

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

```

839 \keys_define:nn { nicematrix / Global }
840 {
841   color-inside .code:n = \@@_err_key_color_inside: ,
842   colortbl-like .code:n = \@@_err_key_color_inside: ,
843   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
844   ampersand-in-blocks .default:n = true ,
845   &-in-blocks .meta:n = ampersand-in-blocks ,
846   no-cell-nodes .code:n =
847     \bool_set_true:N \l_@@_no_cell_nodes_bool
848     \cs_set_protected:Npn \@@_node_cell:
849       { \set@color \box_use_drop:N \l_@@_cell_box } ,
850   no-cell-nodes .value_forbidden:n = true ,
851   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
852   rounded-corners .default:n = 4 pt ,

```

```

853 custom-line .code:n = \@@_custom_line:n { #1 } ,
854 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
855 rules .value_required:n = true ,
856 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
857 standard-cline .default:n = true ,
858 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
859 cell-space-top-limit .value_required:n = true ,
860 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
861 cell-space-bottom-limit .value_required:n = true ,
862 cell-space-limits .meta:n =
863 {
864     cell-space-top-limit = #1 ,
865     cell-space-bottom-limit = #1 ,
866 } ,
867 cell-space-limits .value_required:n = true ,
868 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
869 light-syntax .code:n =
870     \bool_set_true:N \l_@@_light_syntax_bool
871     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
872 light-syntax .value_forbidden:n = true ,
873 light-syntax-expanded .code:n =
874     \bool_set_true:N \l_@@_light_syntax_bool
875     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
876 light-syntax-expanded .value_forbidden:n = true ,
877 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
878 end-of-row .value_required:n = true ,
879 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
880 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
881 last-row .int_set:N = \l_@@_last_row_int ,
882 last-row .default:n = -1 ,
883 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
884 code-for-first-col .value_required:n = true ,
885 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
886 code-for-last-col .value_required:n = true ,
887 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
888 code-for-first-row .value_required:n = true ,
889 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
890 code-for-last-row .value_required:n = true ,
891 hlines .clist_set:N = \l_@@_hlines_clist ,
892 vlines .clist_set:N = \l_@@_vlines_clist ,
893 hlines .default:n = all ,
894 vlines .default:n = all ,
895 vlines-in-sub-matrix .code:n =
896 {
897     \tl_if_single_token:nTF { #1 }
898     {
899         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
900         { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

901     { \cs_set_eq:cN { @@_#1 : } \@@_make_preamble_vlism:n }
902 }
903 { \@@_error:n { One~letter~allowed } }
904 } ,
905 vlines-in-sub-matrix .value_required:n = true ,
906 hvlines .code:n =
907 {
908     \bool_set_true:N \l_@@_hvlines_bool
909     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
910     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
911 } ,
912 hvlines-except-borders .code:n =
913 {
914     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl

```

```

915     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
916     \bool_set_true:N \l_@@_hvlines_bool
917     \bool_set_true:N \l_@@_except_borders_bool
918   } ,
919   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

920   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
921   renew-dots .value_forbidden:n = true ,
922   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
923   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
924   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
925   create-extra-nodes .meta:n =
926     { create-medium-nodes , create-large-nodes } ,
927   left-margin .dim_set:N = \l_@@_left_margin_dim ,
928   left-margin .default:n = \arraycolsep ,
929   right-margin .dim_set:N = \l_@@_right_margin_dim ,
930   right-margin .default:n = \arraycolsep ,
931   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
932   margin .default:n = \arraycolsep ,
933   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
934   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
935   extra-margin .meta:n =
936     { extra-left-margin = #1 , extra-right-margin = #1 } ,
937   extra-margin .value_required:n = true ,
938   respect-arraystretch .code:n =
939     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
940   respect-arraystretch .value_forbidden:n = true ,
941   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
942   pgf-node-code .value_required:n = true
943 }

```

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

```

944 \keys_define:nn { nicematrix / environments }
945 {
946   corners .clist_set:N = \l_@@_corners_clist ,
947   corners .default:n = { NW , SW , NE , SE } ,
948   code-before .code:n =
949     {
950       \tl_if_empty:nF { #1 }
951       {
952         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
953         \bool_set_true:N \l_@@_code_before_bool
954       }
955     } ,
956   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}`.

```

957   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
958   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
959   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
960   baseline .tl_set:N = \l_@@_baseline_tl ,
961   baseline .value_required:n = true ,
962   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)`.

```

963   \str_if_eq:eeTF { #1 } { auto }
964   { \bool_set_true:N \l_@@_auto_columns_width_bool }

```

```

965     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
966     columns-width .value_required:n = true ,
967     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.

```

968     \legacy_if:nF { measuring@ }
969     {
970         \str_set:Nc \l_@@_name_str { #1 }
971         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
972         { \@@_err_duplicate_names:n { #1 } }
973         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
974     } ,
975     name .value_required:n = true ,
976     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
977     code-after .value_required:n = true ,
978 }

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

981 \keys_define:nn { nicematrix / notes }
982 {
983     para .bool_set:N = \l_@@_notes_para_bool ,
984     para .default:n = true ,
985     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
986     code-before .value_required:n = true ,
987     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
988     code-after .value_required:n = true ,
989     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
990     bottomrule .default:n = true ,
991     style .cs_set:Np = \@@_notes_style:n #1 ,
992     style .value_required:n = true ,
993     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
994     label-in-tabular .value_required:n = true ,
995     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
996     label-in-list .value_required:n = true ,
997     enumitem-keys .code:n =
998     {
999         \hook_gput_code:nnn { begindocument } { . }
1000         {
1001             \IfPackageLoadedT { enumitem }
1002             { \setlist* [ tabularnotes ] { #1 } }
1003         }
1004     } ,
1005     enumitem-keys .value_required:n = true ,
1006     enumitem-keys-para .code:n =
1007     {
1008         \hook_gput_code:nnn { begindocument } { . }
1009         {
1010             \IfPackageLoadedT { enumitem }
1011             { \setlist* [ tabularnotes* ] { #1 } }
1012         }
1013     } ,
1014     enumitem-keys-para .value_required:n = true ,
1015     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1016     detect-duplicates .default:n = true ,
1017     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1018 }

1019 \keys_define:nn { nicematrix / delimiters }
1020 {
1021     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1022     max-width .default:n = true ,
1023     color .tl_set:N = \l_@@_delimiters_color_tl ,
1024     color .value_required:n = true ,

```



1025 }

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

```

1026 \keys_define:nn { nicematrix }
1027 {
1028   NiceMatrixOptions .inherit:n =
1029     { nicematrix / Global } ,
1030   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1031   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1032   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1033   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1034   SubMatrix / rules .inherit:n = nicematrix / rules ,
1035   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1036   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1037   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1038   NiceMatrix .inherit:n =
1039     {
1040       nicematrix / Global ,
1041       nicematrix / environments ,
1042     } ,
1043   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1044   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1045   NiceTabular .inherit:n =
1046     {
1047       nicematrix / Global ,
1048       nicematrix / environments
1049     } ,
1050   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1051   NiceTabular / rules .inherit:n = nicematrix / rules ,
1052   NiceTabular / notes .inherit:n = nicematrix / notes ,
1053   NiceArray .inherit:n =
1054     {
1055       nicematrix / Global ,
1056       nicematrix / environments ,
1057     } ,
1058   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1059   NiceArray / rules .inherit:n = nicematrix / rules ,
1060   pNiceArray .inherit:n =
1061     {
1062       nicematrix / Global ,
1063       nicematrix / environments ,
1064     } ,
1065   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1066   pNiceArray / rules .inherit:n = nicematrix / rules ,
1067 }

```

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

```

1068 \keys_define:nn { nicematrix / NiceMatrixOptions }
1069 {
1070   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1071   delimiters / color .value_required:n = true ,
1072   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1073   delimiters / max-width .default:n = true ,
1074   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1075   delimiters .value_required:n = true ,
1076   width .dim_set:N = \l_@@_width_dim ,
1077   width .value_required:n = true ,
1078   last-col .code:n =
1079     \tl_if_empty:nF { #1 }
1080     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }

```

```

1081      \int_zero:N \l_@@_last_col_int ,
1082      small .bool_set:N = \l_@@_small_bool ,
1083      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.

```

1084      renew-matrix .code:n = \@@_renew_matrix: ,
1085      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}`.

```

1086      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.

```

1087      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)`.

```

1088      \str_if_eq:eeTF { #1 } { auto }
1089      { \@@_error:n { Option~auto~for~columns~width } }
1090      { \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.

```

1091      allow-duplicate-names .code:n =
1092      \cs_set:Nn \@@_err_duplicate_names:n { } ,
1093      allow-duplicate-names .value_forbidden:n = true ,
1094      notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1095      notes .value_required:n = true ,
1096      sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1097      sub-matrix .value_required:n = true ,
1098      matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1099      matrix / columns-type .value_required:n = true ,
1100      caption-above .bool_set:N = \l_@@_caption_above_bool ,
1101      caption-above .default:n = true ,
1102      unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1103  }

```

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

```

1104 \NewDocumentCommand \NiceMatrixOptions { m }
1105 { \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.

```

1106 \keys_define:nn { nicematrix / NiceMatrix }
1107 {
1108   last-col .code:n = \tl_if_empty:nTF { #1 }
1109   {
1110     \bool_set_true:N \l_@@_last_col_without_value_bool
1111     \int_set:Nn \l_@@_last_col_int { -1 }
1112   }
1113   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1114   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1115   columns-type .value_required:n = true ,
1116   l .meta:n = { columns-type = l } ,
1117   r .meta:n = { columns-type = r } ,
1118   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

1119   delimiters / color .value_required:n = true ,
1120   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1121   delimiters / max-width .default:n = true ,
1122   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1123   delimiters .value_required:n = true ,
1124   small .bool_set:N = \l_@@_small_bool ,
1125   small .value_forbidden:n = true ,
1126   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1127 }

```

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

```

1128 \keys_define:nn { nicematrix / NiceArray }
1129 {

```

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.

```

1130   small .bool_set:N = \l_@@_small_bool ,
1131   small .value_forbidden:n = true ,
1132   last-col .code:n = \tl_if_empty:nF { #1 }
1133                     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1134                     \int_zero:N \l_@@_last_col_int ,
1135   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1136   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1137   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1138 }

1139 \keys_define:nn { nicematrix / pNiceArray }
1140 {
1141   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1142   last-col .code:n = \tl_if_empty:nF { #1 }
1143                     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1144                     \int_zero:N \l_@@_last_col_int ,
1145   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1146   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1147   delimiters / color .value_required:n = true ,
1148   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1149   delimiters / max-width .default:n = true ,
1150   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1151   delimiters .value_required:n = true ,
1152   small .bool_set:N = \l_@@_small_bool ,
1153   small .value_forbidden:n = true ,
1154   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1155   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1156   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1157 }

```

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

```

1158 \keys_define:nn { nicematrix / NiceTabular }
1159 {

```

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.

```

1160   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1161                     \bool_set_true:N \l_@@_width_used_bool ,
1162   width .value_required:n = true ,
1163   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1164   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1165   tabularnote .value_required:n = true ,
1166   caption .tl_set:N = \l_@@_caption_tl ,
1167   caption .value_required:n = true ,

```

```

1168 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1169 short-caption .value_required:n = true ,
1170 label .tl_set:N = \l_@@_label_tl ,
1171 label .value_required:n = true ,
1172 last-col .code:n = \tl_if_empty:nF { #1 }
1173 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1174 \int_zero:N \l_@@_last_col_int ,
1175 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1176 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1177 unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1178 }

```

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
```

```

1179 \keys_define:nn { nicematrix / CodeAfter }
1180 {
1181   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1182   delimiters / color .value_required:n = true ,
1183   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1184   rules .value_required:n = true ,
1185   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1186   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1187   sub-matrix .value_required:n = true ,
1188   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1189 }

```

## 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}`).

```

1190 \cs_new_protected:Npn \@@_cell_begin:
1191 {

```

`\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.

```
1192   \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).

```
1193   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1194   \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.

```

1195   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1196   {
1197     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1198     { \@@_begin_of_row: }
1199   }

```

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:`.

```
1200 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1201 \@@_tuning_not_tabular_begin:
1202 \@@_tuning_first_row:
1203 \@@_tuning_last_row:
1204 \g_@@_row_style_tl
1205 }
```

The following command will be nullified unless there is a first row.  
Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}
```

We will use a version a little more efficient.

```
1206 \cs_new_protected:Npn \@@_tuning_first_row:
1207 {
1208   \if_int_compare:w \c@iRow = \c_zero_int
1209     \if_int_compare:w \c@jCol > \c_zero_int
1210       \l_@@_code_for_first_row_tl
1211       \xglobal \colorlet { nicematrix-first-row } { . }
1212     \fi:
1213   \fi:
1214 }
```

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:nNnT { \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.

```
1215 \cs_new_protected:Npn \@@_tuning_last_row:
1216 {
1217   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1218     \l_@@_code_for_last_row_tl
1219     \xglobal \colorlet { nicematrix-last-row } { . }
1220   \fi:
1221 }
```

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

```
1222 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```

1223 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1224 {
1225     \m@th
1226     \c_math_toggle_token

```

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

```

1227     \@@_tuning_key_small:
1228 }
1229 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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.

```

1230 \cs_new_protected:Npn \@@_begin_of_row:
1231 {
1232     \int_gincr:N \c@iRow
1233     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1234     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1235     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1236     \pgfpicture
1237     \pgfrememberpicturepositiononpagetrue
1238     \pgfcoordinate
1239     { \@@_env: - row - \int_use:N \c@iRow - base }
1240     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1241     \str_if_empty:NF \l_@@_name_str
1242     {
1243         \pgfnodealias
1244         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1245         { \@@_env: - row - \int_use:N \c@iRow - base }
1246     }
1247     \endpgfpicture
1248 }

```

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.

```

1249 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1250 {
1251     \int_if_zero:nTF { \c@iRow }
1252     {
1253         \dim_compare:nNnT
1254         { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1255         { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1256         \dim_compare:nNnT
1257         { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1258         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1259     }
1260     {
1261         \int_compare:nNnT { \c@iRow } = { \c_one_int }
1262         {
1263             \dim_compare:nNnT
1264             { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1265             { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1266         }
1267     }
1268 }

```

```

1269 \cs_new_protected:Npn \@@_rotate_cell_box:
1270 {
1271   \box_rotate:Nn \l_@@_cell_box { 90 }
1272   \bool_if:NTF \g_@@_rotate_c_bool
1273   {
1274     \hbox_set:Nn \l_@@_cell_box
1275     {
1276       \m@th
1277       \c_math_toggle_token
1278       \vcenter { \box_use:N \l_@@_cell_box }
1279       \c_math_toggle_token
1280     }
1281   }
1282   {
1283     \int_compare:nNtT { \c@iRow } = { \l_@@_last_row_int }
1284     {
1285       \vbox_set_top:Nn \l_@@_cell_box
1286       {
1287         \vbox_to_zero:n { }
1288         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1289         \box_use:N \l_@@_cell_box
1290       }
1291     }
1292   }
1293   \bool_gset_false:N \g_@@_rotate_bool
1294   \bool_gset_false:N \g_@@_rotate_c_bool
1295 }
1296 \cs_new_protected:Npn \@@_adjust_size_box:
1297 {
1298   \dim_compare:nNtT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1299   {
1300     \box_set_wd:Nn \l_@@_cell_box
1301     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1302     \dim_gzero:N \g_@@_blocks_wd_dim
1303   }
1304   \dim_compare:nNtT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1305   {
1306     \box_set_dp:Nn \l_@@_cell_box
1307     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1308     \dim_gzero:N \g_@@_blocks_dp_dim
1309   }
1310   \dim_compare:nNtT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1311   {
1312     \box_set_ht:Nn \l_@@_cell_box
1313     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1314     \dim_gzero:N \g_@@_blocks_ht_dim
1315   }
1316 }
1317 \cs_new_protected:Npn \@@_cell_end:
1318 {

```

The following command is nullified in the tabulars.

```

1319   \@@_tuning_not_tabular_end:
1320   \hbox_set_end:
1321   \@@_cell_end_i:
1322 }
1323 \cs_new_protected:Npn \@@_cell_end_i:
1324 {

```

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.

```

1325   \g_@@_cell_after_hook_tl
1326   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }

```

```

1327 \@@_adjust_size_box:
1328 \box_set_ht:Nn \l_@@_cell_box
1329 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1330 \box_set_dp:Nn \l_@@_cell_box
1331 { \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”).

```

1332 \@@_update_max_cell_width:

```

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

```

1333 \@@_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.

```

1334 \bool_if:NTF \g_@@_empty_cell_bool
1335 { \box_use_drop:N \l_@@_cell_box }
1336 {
1337   \bool_if:NTF \g_@@_not_empty_cell_bool
1338   { \@@_print_node_cell: }
1339   {
1340     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1341     { \@@_print_node_cell: }
1342     { \box_use_drop:N \l_@@_cell_box }
1343   }
1344 }
1345 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1346 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1347 \bool_gset_false:N \g_@@_empty_cell_bool
1348 \bool_gset_false:N \g_@@_not_empty_cell_bool
1349 }

```

The following command will be nullified in our redefinition of `\multicolumn`.

```

1350 \cs_new_protected:Npn \@@_update_max_cell_width:
1351 {
1352   \dim_gset:Nn \g_@@_max_cell_width_dim
1353   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1354 }

```

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`).

```

1355 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1356 {

```



```

1357 \@@_math_toggle:
1358 \hbox_set_end:
1359 \bool_if:NF \g_@@_rotate_bool
1360 {
1361   \hbox_set:Nn \l_@@_cell_box
1362   {
1363     \makebox [ \l_@@_col_width_dim ] [ s ]
1364     { \hbox_unpack_drop:N \l_@@_cell_box }
1365   }
1366 }
1367 \@@_cell_end_i:
1368 }

1369 \pgfset
1370 {
1371   nicematrix / cell-node /.style =
1372   {
1373     inner-sep = \c_zero_dim ,
1374     minimum-width = \c_zero_dim
1375   }
1376 }

```

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.

```

1377 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1378 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1379 {
1380   \use:c
1381   {
1382     __siunitx_table_align_
1383     \bool_if:NTF \l__siunitx_table_text_bool
1384     { \l__siunitx_table_align_text_tl }
1385     { \l__siunitx_table_align_number_tl }
1386   } :n
1387 }
1388 { #1 }
1389 }

```

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!

```

1390 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1391 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1392 {
1393   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1394   \hbox:n
1395   {
1396     \pgfsys@markposition
1397     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1398   }
1399   #1
1400   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1401   \hbox:n
1402   {
1403     \pgfsys@markposition
1404     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1405   }
1406 }

```

```

1407 \cs_new_protected:Npn \@@_print_node_cell:
1408 {
1409   \socket_use:nn { nicematrix / siunitx-wrap }
1410   { \socket_use:nn { nicematrix / create-cell-nodes } \@@_node_cell: }
1411 }

```

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

```

1412 \cs_new_protected:Npn \@@_node_cell:
1413 {
1414   \pgfpicture
1415   \pgfsetbaseline \c_zero_dim
1416   \pgfrememberpicturepositiononpagetrue
1417   \pgfset { nicematrix / cell-node }
1418   \pgfnode
1419   { rectangle }
1420   { base }
1421   {

```

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).

```

1422   \set@color
1423   \box_use:N \l_@@_cell_box
1424 }
1425 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1426 { \l_@@_pgf_node_code_tl }
1427 \str_if_empty:NF \l_@@_name_str
1428 {
1429   \pgfnodealias
1430   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1431   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1432 }
1433 \endpgfpicture
1434 }

```

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}

```

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

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).

```

1435 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1436 {
1437   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1438   { g_@@_ #2 _ lines _ tl }
1439   {
1440     \use:c { @@ _ draw _ #2 : nnn }
1441     { \int_use:N \c@iRow }

```

```

1442         { \int_use:N \c@jCol }
1443         { \exp_not:n { #3 } }
1444     }
1445 }

1446 \cs_new_protected:Npn \@_array:n
1447 {
1448 % \begin{macrocode}
1449 \dim_set:Nn \col@sep
1450 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1451 \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1452 { \def \@halignto { } }
1453 { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1454 \tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

1455 [ \str_if_eq:eeTF \l_@@_baseline_tl { c } { c } { t } ]
1456 }
1457 \cs_generate_variant:Nn \@_array:n { o }

```

We keep in memory the standard version of \ialign because we will redefine \ialign in the environment {NiceArrayWithDelims} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), array uses \ar@ialign instead of \ialign. In that case, of course, you do a saving of \ar@ialign.

```

1458 \bool_if:nTF
1459 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }

```

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.

```

1460 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1461 { \cs_set_eq:NN @@_old_ialign: \ialign }

```

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

```

1462 \cs_new_protected:Npn \@_create_row_node:
1463 {
1464 \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1465 {
1466 \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1467 \@_create_row_node_i:
1468 }
1469 }

1470 \cs_new_protected:Npn \@_create_row_node_i:
1471 {

```

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

```

1472 \hbox
1473 {
1474 \bool_if:NT \l_@@_code_before_bool
1475 {
1476 \vtop
1477 {
1478 \skip_vertical:N 0.5\arrayrulewidth
1479 \pgfsys@markposition
1480 { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1481 \skip_vertical:N -0.5\arrayrulewidth
1482 }
1483 }
1484 \pgfpicture

```

```

1485 \pgfrememberpicturepositiononpagetrue
1486 \pgfcoordinate { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1487 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1488 \str_if_empty:NF \l_@@_name_str
1489 {
1490 \pgfnodealias
1491 { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1492 { \l_@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1493 }
1494 \endpgfpicture
1495 }
1496 }

1497 \cs_new_protected:Npn \l_@@_in_everycr:
1498 {
1499 \bool_if:NT \c_@@_recent_array_bool
1500 {
1501 \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1502 \tbl_update_cell_data_for_next_row:
1503 }
1504 \int_gzero:N \c@jCol
1505 \bool_gset_false:N \g_@@_after_col_zero_bool
1506 \bool_if:NF \g_@@_row_of_col_done_bool
1507 {
1508 \l_@@_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).

```

1509 \clist_if_empty:NF \l_@@_hlines_clist
1510 {
1511 \str_if_eq:eeF \l_@@_hlines_clist { all }
1512 {
1513 \clist_if_in:NiT
1514 \l_@@_hlines_clist
1515 { \int_eval:n { \c@iRow + 1 } }
1516 }
1517 {

```

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.

```

1518 \int_compare:nNnT { \c@iRow } > { -1 }
1519 {
1520 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1521 { \hrule height \arrayrulewidth width \c_zero_dim }
1522 }
1523 }
1524 }
1525 }
1526 }

```

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

```

1527 \cs_set_protected:Npn \l_@@_renew_dots:
1528 {
1529 \cs_set_eq:NN \ldots \l_@@_Ldots:
1530 \cs_set_eq:NN \cdots \l_@@_Cdots:
1531 \cs_set_eq:NN \vdots \l_@@_Vdots:
1532 \cs_set_eq:NN \ddots \l_@@_Ddots:
1533 \cs_set_eq:NN \iddots \l_@@_Iddots:
1534 \cs_set_eq:NN \dots \l_@@_Ldots:
1535 \cs_set_eq:NN \hdotsfor \l_@@_Hdotsfor:
1536 }

```

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>.

```

1537 \hook_gput_code:nnn { begindocument } { . }
1538 {
1539   \IfPackageLoadedTF { booktabs }
1540   {
1541     \cs_new_protected:Npn \@_patch_booktabs:
1542       { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1543   }
1544   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1545 }

```

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`.

```

1546 \cs_new_protected:Npn \@_some_initialization:
1547 {
1548   \@_everycr:
1549   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1550   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1551   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1552   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1553   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1554   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1555 }

```

```

1556 \cs_new_protected:Npn \@_pre_array_ii:
1557 {

```

The total weight of the letters X in the preamble of the array.

```

1558   \fp_gzero:N \g_@@_total_X_weight_fp
1559   \@_expand_clist:N \l_@@_hlines_clist
1560   \@_expand_clist:N \l_@@_vlines_clist
1561   \@_patch_booktabs:
1562   \box_clear_new:N \l_@@_cell_box
1563   \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}`).

```

1564   \bool_if:NT \l_@@_small_bool
1565   {
1566     \def \arraystretch { 0.47 }
1567     \dim_set:Nn \arraycolsep { 1.45 pt }

```

---

<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}`.

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

```

1568     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1569 }

```

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!

```

1570 \bool_if:NT \g_@@_create_cell_nodes_bool
1571 {
1572     \tl_put_right:Nn \@@_begin_of_row:
1573     {
1574         \pgfsys@markposition
1575         { \@@_env: - row - \int_use:N \c@iRow - base }
1576     }
1577     \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1578 }

```

The environment `{array}` (since version 2.6) uses internally the command `\ar@ialign` (and previously, it was `\ialign`). We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1579 \bool_if:nTF
1580 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1581 {
1582     \def \ar@ialign
1583     {
1584         \bool_if:NT \c_@@_testphase_table_bool
1585         \tbl_init_cell_data_for_table:
1586         \@@_some_initialization:
1587         \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this 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`.

```

1588         \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1589         \halign
1590     }
1591 }

```

The following part should be deleted when we will delete the boolean `\c_@@_recent_array_bool` (when we consider the version 2.6a of `array` is required). Moreover, `revtex4-2` modifies `array` and provides commands which are meant to be the standard version of `array` but, at the date of november 2024, these commands corresponds to the *old* version of `array`, that is to say without the `\ar@ialign`.

```

1592 {
1593     \def \ialign
1594     {
1595         \@@_some_initialization:
1596         \dim_zero:N \tabskip
1597         \cs_set_eq:NN \ialign \@@_old_ialign:
1598         \halign
1599     }
1600 }

```

It seems that there is a problem when `nicematrix` is used with in `revtex4-2` with the package `colortbl` loaded. The following code prevent that problem but it does *not* treat the actual problem! It’s only a patch *ad hoc*.

That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```

1601 \bool_if:NT \c_@@_revtex_bool
1602 {
1603     \IfPackageLoadedT { colortbl }
1604     { \cs_set_protected:Npn \CT@setup { } }
1605 }

```

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).

```

1606 \cs_set_eq:NN \@@_old_ldots: \ldots
1607 \cs_set_eq:NN \@@_old_cdots: \cdots
1608 \cs_set_eq:NN \@@_old_vdots: \vdots
1609 \cs_set_eq:NN \@@_old_ddots: \ddots
1610 \cs_set_eq:NN \@@_old_iddots: \iddots
1611 \bool_if:NTF \l_@@_standard_cline_bool
1612 { \cs_set_eq:NN \cline \@@_standard_cline: }
1613 { \cs_set_eq:NN \cline \@@_cline: }
1614 \cs_set_eq:NN \Ldots \@@_Ldots:
1615 \cs_set_eq:NN \Cdots \@@_Cdots:
1616 \cs_set_eq:NN \Vdots \@@_Vdots:
1617 \cs_set_eq:NN \Ddots \@@_Ddots:
1618 \cs_set_eq:NN \Iddots \@@_Iddots:
1619 \cs_set_eq:NN \Hline \@@_Hline:
1620 \cs_set_eq:NN \Hspace \@@_Hspace:
1621 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1622 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1623 \cs_set_eq:NN \Block \@@_Block:
1624 \cs_set_eq:NN \rotate \@@_rotate:
1625 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1626 \cs_set_eq:NN \dotfill \@@_dotfill:
1627 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1628 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1629 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1630 \cs_set_eq:NN \TopRule \@@_TopRule
1631 \cs_set_eq:NN \MidRule \@@_MidRule
1632 \cs_set_eq:NN \BottomRule \@@_BottomRule
1633 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1634 \cs_set_eq:NN \Hbrace \@@_Hbrace
1635 \cs_set_eq:NN \Vbrace \@@_Vbrace
1636 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1637 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1638 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1639 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1640 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1641 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1642 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1643 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1644 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1645 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1646 \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:`.

```

1647 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1648 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1649 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1650 \@@_revert_colortbl:

```

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

```

1651 \tl_if_exist:NT \l_@@_note_in_caption_tl
1652 {
1653   \tl_if_empty:NF \l_@@_note_in_caption_tl
1654   {
1655     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl

```

```

1656         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1657     }
1658 }

```

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).

```

1659     \seq_gclear:N \g_@@_multicolumn_cells_seq
1660     \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

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

```

1661     \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 or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1662     \int_gzero_new: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.

```

1663     \int_gzero_new:N \g_@@_col_total_int
1664     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1665     \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.

```

1666     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1667     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1668     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1669     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1670     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1671     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1672     \tl_gclear:N \g_nicematrix_code_before_tl
1673     \tl_gclear:N \g_@@_pre_code_before_tl
1674 }

```

This is the end of `\@@_pre_array_ii:`.

The command `\@@_pre_array:` will be executed after analyse of the keys of the environment.

```

1675 \cs_new_protected:Npn \@@_pre_array:
1676 {
1677     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1678     \int_gzero_new:N \c@iRow
1679     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1680     \int_gzero_new:N \c@jCol

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_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-column` (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).

```

1681     \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1682     {
1683         \bool_set_true:N \l_@@_last_row_without_value_bool
1684         \bool_if:NT \g_@@_aux_found_bool
1685             { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1686     }

```



```

1687 \int_compare:nNt { \l_@@_last_col_int } = { -1 }
1688 {
1689     \bool_if:NT \g_@@_aux_found_bool
1690     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1691 }

```

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”.

```

1692 \int_compare:nNt { \l_@@_last_row_int } > { -2 }
1693 {
1694     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1695     {
1696         \dim_compare:nNt { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1697         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1698         \dim_compare:nNt { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1699         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1700     }
1701 }

1702 \seq_gclear:N \g_@@_cols_vlism_seq
1703 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore.`

```

1704 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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 clear that variable because it will be reconstructed during the creation of the array.

```

1705 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1706 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1707 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1708 \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.

```

1709 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1710 \@@_pre_array_ii:

```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```

1711 \box_clear_new:N \l_@@_the_array_box

```

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]`).

```

1712 \dim_zero_new:N \l_@@_left_delim_dim
1713 \dim_zero_new:N \l_@@_right_delim_dim
1714 \bool_if:NTF \g_@@_delims_bool
1715 {

```

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

```

1716     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1717     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1718     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1719     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1720   }
1721   {
1722     \dim_gset:Nn \l_@@_left_delim_dim
1723       { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1724     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1725   }

```

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 `\c_math_toggle_token` also).

```

1726     \hbox_set:Nw \l_@@_the_array_box
1727     \skip_horizontal:N \l_@@_left_margin_dim
1728     \skip_horizontal:N \l_@@_extra_left_margin_dim
1729     \bool_if:NT \c_@@_recent_array_bool
1730     { \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).

```

1731     \m@th
1732     \c_math_toggle_token
1733     \bool_if:NTF \l_@@_light_syntax_bool
1734       { \use:c { @@-light-syntax } }
1735       { \use:c { @@-normal-syntax } }
1736   }

```

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

```

1737 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1738   {
1739     \tl_set:Nn \l_tmpa_tl { #1 }
1740     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1741       { \@@_rescan_for_spanish:N \l_tmpa_tl }
1742     \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1743     \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...

```

1744     \@@_pre_array:
1745   }

```

## 9 The `\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).

```

1746 \cs_new_protected:Npn \@@_pre_code_before:
1747   {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in 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).

```

1748 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1749 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1750 \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1751 \int_set:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }

```

Now, 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.

```

1752 \pgfsys@markposition { \@@_env: - position }
1753 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1754 \pgfpicture
1755 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1756 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1757 {
1758   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1759   \pgfcoordinate { \@@_env: - row - ##1 }
1760   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1761 }

```

Now, the recreation of the col nodes.

```

1762 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1763 {
1764   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1765   \pgfcoordinate { \@@_env: - col - ##1 }
1766   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1767 }

```

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

```

1768 \@@_create_diag_nodes:

```

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

```

1769 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1770 \endpgfpicture

```

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

```

1771 \@@_create_blocks_nodes:
1772 \IfPackageLoadedT { tikz }
1773 {
1774   \tikzset
1775   {
1776     every-picture / .style =
1777     { overlay , name~prefix = \@@_env: - }
1778   }
1779 }
1780 \cs_set_eq:NN \cellcolor \@@_cellcolor
1781 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1782 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1783 \cs_set_eq:NN \rowcolor \@@_rowcolor
1784 \cs_set_eq:NN \rowcolors \@@_rowcolors
1785 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1786 \cs_set_eq:NN \arraycolor \@@_arraycolor
1787 \cs_set_eq:NN \columncolor \@@_columncolor
1788 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1789 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1790 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1791 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1792 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1793 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1794 }

```

```

1795 \cs_new_protected:Npn \@@_exec_code_before:
1796 {

```

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 detected whether we actually have coloring instructions to execute...

```

1797 \clist_map_inline:Nn \l_@@_corners_cells_clist
1798 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1799 \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`.

```

1800 \@@_add_to_colors_seq:nn { { nocolor } } { }
1801 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1802 \group_begin:

```

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

```

1803 \if_mode_math:
1804 \@@_exec_code_before_i:
1805 \else:
1806 \c_math_toggle_token
1807 \@@_exec_code_before_i:
1808 \c_math_toggle_token
1809 \fi:
1810 \group_end:
1811 }

```

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`).

```

1812 \cs_new_protected:Npn \@@_exec_code_before_i:
1813 {
1814 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1815 { \@@_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`.

```

1816 \exp_last_unbraced:No \@@_CodeBefore_keys:
1817 \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.

```

1818 \@@_actually_color:
1819 \l_@@_code_before_tl
1820 \q_stop
1821 }

```

```

1822 \keys_define:nn { nicematrix / CodeBefore }
1823 {
1824 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1825 create-cell-nodes .default:n = true ,
1826 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1827 sub-matrix .value_required:n = true ,

```

```

1828     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1829     delimiters / color .value_required:n = true ,
1830     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1831 }
1832 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1833 {
1834     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1835     \@@_CodeBefore:w
1836 }

```

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.

```

1837 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1838 {
1839     \bool_if:NT \g_@@_aux_found_bool
1840     {
1841         \@@_pre_code_before:
1842         \legacy_if:nF { measuring@ } { #1 }
1843     }
1844 }

```

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.

```

1845 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1846 {
1847     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1848     {
1849         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1850         \pgfcoordinate { \@@_env: - row - ##1 - base }
1851         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1852         \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1853         {
1854             \cs_if_exist:cT
1855             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1856             {
1857                 \pgfsys@getposition
1858                 { \@@_env: - ##1 - #####1 - NW }
1859                 \@@_node_position:
1860                 \pgfsys@getposition
1861                 { \@@_env: - ##1 - #####1 - SE }
1862                 \@@_node_position_i:
1863                 \@@_pgf_rect_node:nnn
1864                 { \@@_env: - ##1 - #####1 }
1865                 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1866                 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1867             }
1868         }
1869     }
1870     \@@_create_extra_nodes:
1871     \@@_create_aliases_last:
1872 }
1873 \cs_new_protected:Npn \@@_create_aliases_last:
1874 {
1875     \int_step_inline:nn { \c@iRow }
1876     {
1877         \pgfnodealias
1878         { \@@_env: - ##1 - last }
1879         { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1880     }
1881     \int_step_inline:nn { \c@jCol }
1882     {
1883         \pgfnodealias
1884         { \@@_env: - last - ##1 }
1885         { \@@_env: - \int_use:N \c@iRow - ##1 }
1886     }
1887     \pgfnodealias % added 2025-04-05
1888     { \@@_env: - last - last }
1889     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1890 }

```

```

1891 \cs_new_protected:Npn \@@_create_blocks_nodes:
1892 {
1893     \pgfpicture
1894     \pgf@relevantforpicturesizefalse
1895     \pgfrememberpicturepositiononpagetrue
1896     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1897     { \@@_create_one_block_node:nnnnn ##1 }
1898     \endpgfpicture
1899 }

```

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>

```

1900 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1901 {
1902     \tl_if_empty:nF { #5 }
1903     {
1904         \@@_qpoint:n { col - #2 }
1905         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1906         \@@_qpoint:n { #1 }
1907         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1908         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1909         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1910         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1911         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1912         \@@_pgf_rect_node:nnnnn
1913         { \@@_env: - #5 }
1914         { \dim_use:N \l_tmpa_dim }
1915         { \dim_use:N \l_tmpb_dim }
1916         { \dim_use:N \l_@@_tmpc_dim }
1917         { \dim_use:N \l_@@_tmpd_dim }
1918     }
1919 }

```

```

1920 \cs_new_protected:Npn \@@_patch_for_revtext:
1921 {
1922     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1923     \cs_set_eq:NN \@array \@array@array
1924     \cs_set_eq:NN \@tabular \@tabular@array
1925     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1926     \cs_set_eq:NN \array \array@array
1927     \cs_set_eq:NN \endarray \endarray@array
1928     \cs_set:Npn \endtabular { \endarray $\egroup } % $
1929     \cs_set_eq:NN \@mkpream \@mkpream@array
1930     \cs_set_eq:NN \@classx \@classx@array
1931     \cs_set_eq:NN \insert@column \insert@column@array
1932     \cs_set_eq:NN \@arraycr \@arraycr@array

```

<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).

```

1933 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1934 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1935 }

```

## 10 The environment `{NiceArrayWithDelims}`

```

1936 \NewDocumentEnvironment { NiceArrayWithDelims }
1937 { m m O { } m ! O { } t \CodeBefore }
1938 {
1939 \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1940 \@@_provide_pgfsyspdfmark:
1941 \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.

```

1942 \bgroup

1943 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1944 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1945 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1946 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1947 \int_gzero:N \g_@@_block_box_int
1948 \dim_gzero:N \g_@@_width_last_col_dim
1949 \dim_gzero:N \g_@@_width_first_col_dim
1950 \bool_gset_false:N \g_@@_row_of_col_done_bool
1951 \str_if_empty:NT \g_@@_name_env_str
1952 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1953 \bool_if:NTF \l_@@_tabular_bool
1954 { \mode_leave_vertical: }
1955 { \@@_test_if_math_mode: }
1956 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1957 \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.

```

1958 \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.

```

1959 \cs_if_exist:NT \tikz@library@external@loaded
1960 {
1961 \tikzexternaldisable
1962 \cs_if_exist:NT \ifstandalone
1963 { \tikzset { external / optimize = false } }
1964 }

```

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

```

1965 \int_gincr:N \g_@@_env_int
1966 \bool_if:NF \l_@@_block_auto_columns_width_bool
1967 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

---

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

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.

```
1968 \seq_gclear:N \g_@@_blocks_seq
1969 \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`.

```
1970 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1971 \seq_gclear:N \g_@@_pos_of_xdots_seq
1972 \tl_gclear_new:N \g_@@_code_before_tl
1973 \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.

```
1974 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1975 {
1976   \bool_gset_true:N \g_@@_aux_found_bool
1977   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1978 }
1979 { \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.

```
1980 \tl_gclear:N \g_@@_aux_tl
1981 \tl_if_empty:NF \g_@@_code_before_tl
1982 {
1983   \bool_set_true:N \l_@@_code_before_bool
1984   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1985 }
1986 \tl_if_empty:NF \g_@@_pre_code_before_tl
1987 { \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`.

```
1988 \bool_if:NTF \g_@@_delims_bool
1989 { \keys_set:nn { nicematrix / pNiceArray } }
1990 { \keys_set:nn { nicematrix / NiceArray } }
1991 { #3 , #5 }
```

```
1992 \@@_set_CTarc:o \l_@@_rules_color_tl
```

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:.`

```
1993 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1994 }
```

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

```
1995 {
1996   \bool_if:NTF \l_@@_light_syntax_bool
1997   { \use:c { end @@-light-syntax } }
1998   { \use:c { end @@-normal-syntax } }
1999   \c_math_toggle_token
2000   \skip_horizontal:N \l_@@_right_margin_dim
2001   \skip_horizontal:N \l_@@_extra_right_margin_dim
2002
2003   % awful workaround
2004   \int_if_zero:nT { \g_@@_col_total_int }
2005   {
```



```

2006      \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim }
2007      {
2008        \skip_horizontal:n { - \l_@@_columns_width_dim }
2009        \bool_if:NTF \l_@@_tabular_bool
2010        { \skip_horizontal:n { - 2 \tabcolsep } }
2011        { \skip_horizontal:n { - 2 \arraycolsep } }
2012      }
2013    }
2014    \hbox_set_end:
2015    \bool_if:NT \c_@@_recent_array_bool
2016    { \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.

```

2017    \bool_if:NT \l_@@_width_used_bool
2018    {
2019      \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2020      { \@@_error_or_warning:n { width-without-X-columns } }
2021    }

```

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  $x$ .

```

2022    \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2023    { \@@_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).

```

2024    \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2025    {
2026      \bool_if:NF \l_@@_last_row_without_value_bool
2027      {
2028        \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2029        {
2030          \@@_error:n { Wrong-last-row }
2031          \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2032        }
2033      }
2034    }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` change: `\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>

```

2035    \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2036    \bool_if:NTF \g_@@_last_col_found_bool
2037    { \int_gdecr:N \c@jCol }
2038    {
2039      \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2040      { \@@_error:n { last-col-not-used } }
2041    }

```

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

```

2042    \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2043    \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2044    { \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. 91).

---

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

```

2045 \int_if_zero:nT { \l_@@_first_col_int }
2046 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2047 \bool_if:nTF { ! \g_@@_delims_bool }
2048 {
2049   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2050   { \@@_use_arraybox_with_notes_c: }
2051   {
2052     \str_if_eq:eeTF \l_@@_baseline_tl { b }
2053     { \@@_use_arraybox_with_notes_b: }
2054     { \@@_use_arraybox_with_notes: }
2055   }
2056 }

```

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).

```

2057 {
2058   \int_if_zero:nTF { \l_@@_first_row_int }
2059   {
2060     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2061     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2062   }
2063   { \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>

```

2064 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2065 {
2066   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2067   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2068 }
2069 { \dim_zero:N \l_tmpb_dim }
2070 \hbox_set:Nn \l_tmpa_box
2071 {
2072   \m@th
2073   \c_math_toggle_token
2074   \@@_color:o \l_@@_delimiters_color_tl
2075   \exp_after:wN \left \g_@@_left_delim_tl
2076   \vcenter
2077   {

```

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.

```

2078   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2079   \hbox
2080   {
2081     \bool_if:NTF \l_@@_tabular_bool
2082     { \skip_horizontal:n { - \tabcolsep } }
2083     { \skip_horizontal:n { - \arraycolsep } }
2084     \@@_use_arraybox_with_notes_c:
2085     \bool_if:NTF \l_@@_tabular_bool
2086     { \skip_horizontal:n { - \tabcolsep } }
2087     { \skip_horizontal:n { - \arraycolsep } }
2088   }

```

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

```

2089   \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2090   }
2091   \exp_after:wN \right \g_@@_right_delim_tl
2092   \c_math_toggle_token
2093   }

```

---

<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).

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.

```

2094     \bool_if:NTF \l_@@_delimiters_max_width_bool
2095     {
2096         \@@_put_box_in_flow_bis:nn
2097         { \g_@@_left_delim_tl }
2098         { \g_@@_right_delim_tl }
2099     }
2100     \@@_put_box_in_flow:
2101 }

```

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. 92).

```

2102     \bool_if:NT \g_@@_last_col_found_bool
2103     { \skip_horizontal:N \g_@@_width_last_col_dim }
2104     \bool_if:NT \l_@@_preamble_bool
2105     {
2106         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2107         { \@@_err_columns_not_used: }
2108     }
2109     \@@_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.

```

2110     \egroup

```

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

```

2111     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2112     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2113     \iow_now:Ne \@mainaux
2114     {
2115         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2116         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2117         { \exp_not:o \g_@@_aux_tl }
2118     }
2119     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2120     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2121 }

```

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

```

2122 \cs_new_protected:Npn \@@_err_columns_not_used:
2123 {
2124     \@@_warning:n { columns~not~used }
2125     \cs_gset:Npn \@@_err_columns_not_used: { }
2126 }

```

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  $x$ .

```

2127 \cs_new_protected:Npn \@@_compute_width_X:
2128 {
2129     \tl_gput_right:Ne \g_@@_aux_tl
2130     {
2131         \bool_set_true:N \l_@@_X_columns_aux_bool
2132         \dim_set:Nn \l_@@_X_columns_dim
2133         {
2134             \dim_compare:nNnTF
2135             {
2136                 \dim_abs:n

```

```

2137         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2138     }
2139     <
2140     { 0.001 pt }
2141     { \dim_use:N \l_@@_X_columns_dim }
2142     {
2143         \dim_eval:n
2144         {
2145             \fp_to_dim:n
2146             {
2147                 (
2148                     \dim_eval:n
2149                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2150                 )
2151                 / \fp_use:N \g_@@_total_X_weight_fp
2152             }
2153             + \l_@@_X_columns_dim
2154         }
2155     }
2156 }
2157 }
2158 }

```

## 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` also.

```

2159 \cs_new_protected:Npn \@@_transform_preamble:
2160 {
2161     \@@_transform_preamble_i:
2162     \@@_transform_preamble_ii:
2163 }
2164 \cs_new_protected:Npn \@@_transform_preamble_i:
2165 {
2166     \int_gzero:N \c@jCol

```

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

```

2167     \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.

```

2168     \bool_gset_false:N \g_tmpb_bool

```

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

```

2169     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2170     \int_zero:N \l_tmpa_int
2171     \tl_gclear:N \g_@@_array_preamble_tl
2172     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2173     {
2174         \tl_gset:Nn \g_@@_array_preamble_tl
2175         { ! { \skip_horizontal:N \arrayrulewidth } }
2176     }
2177     {
2178         \clist_if_in:NnT \l_@@_vlines_clist 1

```

```

2179     {
2180         \tl_gset:Nn \g_@@_array_preamble_tl
2181         { ! { \skip_horizontal:N \arrayrulewidth } }
2182     }
2183 }

```

Now, we actually make the preamble (which will be given to {array}). It will be stored in \g\_@@\_array\_preamble\_tl.

```

2184     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2185     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2186     \@@_replace_columncolor:
2187 }

```

```

2188 \cs_new_protected:Npn \@@_transform_preamble_ii:
2189 {

```

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

```

2190     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2191     {
2192         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2193         { \bool_gset_true:N \g_@@_delims_bool }
2194     }
2195     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2196     \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).

```

2197     \int_if_zero:nTF { \l_@@_first_col_int }
2198     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2199     {
2200         \bool_if:NF \g_@@_delims_bool
2201         {
2202             \bool_if:NF \l_@@_tabular_bool
2203             {
2204                 \clist_if_empty:NT \l_@@_vlines_clist
2205                 {
2206                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2207                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2208                 }
2209             }
2210         }
2211     }
2212     \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2213     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2214     {
2215         \bool_if:NF \g_@@_delims_bool
2216         {
2217             \bool_if:NF \l_@@_tabular_bool
2218             {
2219                 \clist_if_empty:NT \l_@@_vlines_clist
2220                 {
2221                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2222                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2223                 }
2224             }
2225         }
2226     }

```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it's not possible to do that in `{NiceTabular*}` (we control that with the value of `\l_@@_tabular_width_dim`).

```
2227 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2228 {
```

If the tagging of the tabulars is done (part of the Tagging Project), you don't activate that mechanism because it would create a dummy column of tagged empty cells.

```
2229 \bool_if:NF \c_@@_testphase_table_bool
2230 {
2231   \tl_gput_right:Nn \g_@@_array_preamble_tl
2232     { > { \@@_error_too_much_cols: } 1 }
2233 }
2234 }
2235 }
```

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.

```
2236 \cs_new_protected:Npn \@@_rec_preamble:n #1
2237 {
```

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>

```
2238 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2239 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2240 {
```

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

```
2241 \cs_if_exist:cTF { NC @ find @ #1 }
2242 {
2243   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2244   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2245 }
2246 {
2247   \str_if_eq:nnTF { #1 } { S }
2248   { \@@_fatal:n { unknown~column~type~S } }
2249   { \@@_fatal:nn { unknown~column~type } { #1 } }
2250 }
2251 }
2252 }
```

For c, l and r

```
2253 \cs_new_protected:Npn \@@_c: #1
2254 {
2255   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2256   \tl_gclear:N \g_@@_pre_cell_tl
2257   \tl_gput_right:Nn \g_@@_array_preamble_tl
2258     { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2259 \int_gincr:N \c@jCol
2260 \@@_rec_preamble_after_col:n
2261 }
```

---

<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`.

```

2262 \cs_new_protected:Npn \@@_l: #1
2263 {
2264   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2265   \tl_gclear:N \g_@@_pre_cell_tl
2266   \tl_gput_right:Nn \g_@@_array_preamble_tl
2267   {
2268     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2269     l
2270     < \@@_cell_end:
2271   }
2272   \int_gincr:N \c@jCol
2273   \@@_rec_preamble_after_col:n
2274 }
2275 \cs_new_protected:Npn \@@_r: #1
2276 {
2277   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2278   \tl_gclear:N \g_@@_pre_cell_tl
2279   \tl_gput_right:Nn \g_@@_array_preamble_tl
2280   {
2281     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2282     r
2283     < \@@_cell_end:
2284   }
2285   \int_gincr:N \c@jCol
2286   \@@_rec_preamble_after_col:n
2287 }

```

For ! and @

```

2288 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2289 {
2290   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2291   \@@_rec_preamble:n
2292 }
2293 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2294 \cs_new_protected:cpn { @@ _ | : } #1
2295 {

```

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

```

2296   \int_incr:N \l_tmpa_int
2297   \@@_make_preamble_i_i:n
2298 }
2299 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2300 {
2301   \str_if_eq:nnTF { #1 } { | }
2302   { \use:c { @@ _ | : } | }
2303   { \@@_make_preamble_i_ii:nn { } #1 }
2304 }
2305 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2306 {
2307   \str_if_eq:nnTF { #2 } { [ ] }
2308   { \@@_make_preamble_i_iii:nw { #1 } [ ] }
2309   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2310 }
2311 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2312 { \@@_make_preamble_i_iii:nn { #1 , #2 } }
2313 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2314 {
2315   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2316   \tl_gput_right:Ne \g_@@_array_preamble_tl
2317   {

```

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

```

2318     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2319   }
2320   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2321   {
2322     \@@_vline:n
2323     {
2324       position = \int_eval:n { \c@jCol + 1 } ,
2325       multiplicity = \int_use:N \l_tmpa_int ,
2326       total-width = \dim_use:N \l_@@_rule_width_dim ,
2327       #2
2328     }

```

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.

```

2329   }
2330   \int_zero:N \l_tmpa_int
2331   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2332   \@@_rec_preamble:n #1
2333 }

2334 \cs_new_protected:cpn { @@_ > : } #1 #2
2335 {
2336   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2337   \@@_rec_preamble:n
2338 }

2339 \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.

```

2340 \keys_define:nn { nicematrix / p-column }
2341 {
2342   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2343   r .value_forbidden:n = true ,
2344   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2345   c .value_forbidden:n = true ,
2346   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2347   l .value_forbidden:n = true ,
2348   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2349   S .value_forbidden:n = true ,
2350   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2351   p .value_forbidden:n = true ,
2352   t .meta:n = p ,
2353   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2354   m .value_forbidden:n = true ,
2355   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2356   b .value_forbidden:n = true
2357 }

```

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

```

2358 \cs_new_protected:Npn \@@_p: #1
2359 {
2360   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2361   \@@_make_preamble_ii_i:n
2362 }
2363 \cs_set_eq:NN \@@_b: \@@_p:
2364 \cs_set_eq:NN \@@_m: \@@_p:

```



```

2365 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2366 {
2367   \str_if_eq:nnTF { #1 } { [ ] }
2368   { \@@_make_preamble_ii_ii:w [ ] }
2369   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2370 }
2371 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2372 { \@@_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.

```

2373 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2374 {

```

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).

```

2375   \str_set:Nn \l_@@_hpos_col_str { j }
2376   \@@_keys_p_column:n { #1 }
2377   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2378 }
2379 \cs_new_protected:Npn \@@_keys_p_column:n #1
2380 { \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.

```

2381 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2382 {
2383   \use:e
2384   {
2385     \@@_make_preamble_ii_v:nnnnnnnn
2386     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2387     { \dim_eval:n { #1 } }
2388     {

```

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.

```

2389       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2390       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2391       {

```

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

```

2392         \def \exp_not:N \l_@@_hpos_cell_tl
2393         { \str_lowercase:f { \l_@@_hpos_col_str } }
2394       }
2395       \IfPackageLoadedTF { ragged2e }
2396       {
2397         \str_case:on \l_@@_hpos_col_str
2398         {

```

The following `\exp_not:N` are mandatory.

```

2399           c { \exp_not:N \Centering }
2400           l { \exp_not:N \RaggedRight }
2401           r { \exp_not:N \RaggedLeft }
2402         }
2403       }
2404       {
2405         \str_case:on \l_@@_hpos_col_str
2406         {
2407           c { \exp_not:N \centering }
2408           l { \exp_not:N \raggedright }
2409           r { \exp_not:N \raggedleft }

```

```

2410     }
2411   }
2412   #3
2413 }
2414 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2415 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2416 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2417 { #2 }
2418 {
2419   \str_case:onF \l_@@_hpos_col_str
2420   {
2421     { j } { c }
2422     { si } { c }
2423   }

```

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

```

2424     { \str_lowercase:f \l_@@_hpos_col_str }
2425   }
2426 }

```

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

```

2427   \int_gincr:N \c@jCol
2428   \@@_rec_preamble_after_col:n
2429 }

```

**#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*.

```

2430 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2431 {
2432   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2433   {
2434     \tl_gput_right:Nn \g_@@_array_preamble_tl
2435     { > \@@_test_if_empty_for_S: }
2436   }
2437   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2438   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2439   \tl_gclear:N \g_@@_pre_cell_tl
2440   \tl_gput_right:Nn \g_@@_array_preamble_tl
2441   {
2442     > {

```

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.

```

2443   \dim_set:Nn \l_@@_col_width_dim { #2 }
2444   \bool_if:NT \c_@@_testphase_table_bool
2445   { \tag_struct_begin:n { tag = Div } }
2446   \@@_cell_begin:

```

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

```

2447   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2448     \everypar
2449     {
2450         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2451         \everypar { }
2452     }
2453     \bool_if:NT \c_@@_testphase_table_bool { \tagpdfparaOn }

```

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

```

2454     #3

```

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

```

2455     \g_@@_row_style_tl
2456     \arraybackslash
2457     #5
2458 }
2459 #8
2460 < {
2461     #6

```

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

```

2462     \@finalstrut \@arstrutbox
2463     \use:c { end #7 }

```

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

```

2464     #4
2465     \@@_cell_end:
2466     \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2467 }
2468 }
2469 }

```

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

```

2470 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2471 {

```

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 `&`).

```

2472     \group_align_safe_begin:
2473     \peek_meaning:NTF &
2474     { \@@_the_cell_is_empty: }
2475     {
2476         \peek_meaning:NTF \\\
2477         { \@@_the_cell_is_empty: }
2478         {
2479             \peek_meaning:NTF \crcr
2480             \@@_the_cell_is_empty:
2481             \group_align_safe_end:
2482         }
2483     }
2484 }

2485 \cs_new_protected:Npn \@@_the_cell_is_empty:
2486 {
2487     \group_align_safe_end:
2488     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2489     {

```

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

```

2490     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2491     \skip_horizontal:N \l_@@_col_width_dim

```

```

2492     }
2493   }
2494   \cs_new_protected:Npn \@@_test_if_empty_for_S:
2495   {
2496     \peek_meaning:NT \__siunitx_table_skip:n
2497     { \bool_gset_true:N \g_@@_empty_cell_bool }
2498   }

```

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

```

2499   \cs_new_protected:Npn \@@_center_cell_box:
2500   {

```

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

```

2501     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2502     {
2503       \dim_compare:nNnT
2504       { \box_ht:N \l_@@_cell_box }
2505       >

```

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).

```

2506         { \box_ht:N \strutbox }
2507         {
2508           \hbox_set:Nn \l_@@_cell_box
2509           {
2510             \box_move_down:nn
2511             {
2512               ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2513                 + \baselineskip ) / 2
2514             }
2515             { \box_use:N \l_@@_cell_box }
2516           }
2517         }
2518       }
2519     }

```

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

```

2520   \cs_new_protected:Npn \@@_V: #1 #2
2521   {
2522     \str_if_eq:nnTF { #1 } { [ ] }
2523     { \@@_make_preamble_V_i:w [ ] }
2524     { \@@_make_preamble_V_i:w [ ] { #2 } }
2525   }
2526   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2527   { \@@_make_preamble_V_ii:nn { #1 } }
2528   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2529   {
2530     \str_set:Nn \l_@@_vpos_col_str { p }
2531     \str_set:Nn \l_@@_hpos_col_str { j }
2532     \@@_keys_p_column:n { #1 }
2533     \IfPackageLoadedTF { varwidth }
2534     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2535     {
2536       \@@_error_or_warning:n { varwidth-not-loaded }
2537       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2538     }
2539   }

```

For `w` and `W`

```

2540 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2541 \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.

2542 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2543 {
2544   \str_if_eq:nnTF { #3 } { s }
2545   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2546   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2547 }

```

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.

```

2548 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2549 {
2550   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2551   \tl_gclear:N \g_@@_pre_cell_tl
2552   \tl_gput_right:Nn \g_@@_array_preamble_tl
2553   {
2554     > {
2555       \dim_set:Nn \l_@@_col_width_dim { #2 }
2556       \@@_cell_begin:
2557       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2558     }
2559     c
2560     < {
2561       \@@_cell_end_for_w_s:
2562       #1
2563       \@@_adjust_size_box:
2564       \box_use_drop:N \l_@@_cell_box
2565     }
2566   }
2567   \int_gincr:N \c_jCol
2568   \@@_rec_preamble_after_col:n
2569 }

```

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

```

2570 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2571 {
2572   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2573   \tl_gclear:N \g_@@_pre_cell_tl
2574   \tl_gput_right:Nn \g_@@_array_preamble_tl
2575   {
2576     > {

```

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.

```

2577       \dim_set:Nn \l_@@_col_width_dim { #4 }
2578       \hbox_set:Nw \l_@@_cell_box
2579       \@@_cell_begin:
2580       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2581     }
2582     c
2583     < {
2584       \@@_cell_end:
2585       \hbox_set_end:
2586       #1

```

```

2587         \@@_adjust_size_box:
2588         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2589     }
2590 }

```

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

```

2591     \int_gincr:N \c@jCol
2592     \@@_rec_preamble_after_col:n
2593 }

```

```

2594 \cs_new_protected:Npn \@@_special_W:
2595 {
2596     \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2597     { \@@_warning:n { W~warning } }
2598 }

```

For S (of siunitx).

```

2599 \cs_new_protected:Npn \@@_S: #1 #2
2600 {
2601     \str_if_eq:nnTF { #2 } { [ ] }
2602     { \@@_make_preamble_S:w [ ] }
2603     { \@@_make_preamble_S:w [ ] { #2 } }
2604 }
2605 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2606 { \@@_make_preamble_S_i:n { #1 } }
2607 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2608 {
2609     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2610     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2611     \tl_gclear:N \g_@@_pre_cell_tl
2612     \tl_gput_right:Nn \g_@@_array_preamble_tl
2613     {
2614         > {

```

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).

```

2615         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2616         \keys_set:nn { siunitx } { #1 }
2617         \@@_cell_begin:
2618         \siunitx_cell_begin:w
2619     }
2620     c
2621     <
2622     {
2623         \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`).

```

2624         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2625         {
2626             \bool_if:NTF \l__siunitx_table_text_bool
2627             { \bool_set_true:N }
2628             { \bool_set_false:N }
2629             \l__siunitx_table_text_bool
2630         }
2631         \@@_cell_end:
2632     }
2633 }

```

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

```

2634 \int_gincr:N \c@jCol
2635 \@@_rec_preamble_after_col:n
2636 }

```

For (, [ and \{.

```

2637 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2638 {
2639 \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.

```

2640 \int_if_zero:nTF { \c@jCol }
2641 {
2642 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2643 {

```

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

```

2644 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2645 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2646 \@@_rec_preamble:n #2
2647 }
2648 {
2649 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2650 \@@_make_preamble_iv:nn { #1 } { #2 }
2651 }
2652 }
2653 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2654 }
2655 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2656 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2657 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2658 {
2659 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2660 { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2661 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2662 {
2663 \@@_error:nn { delimiter~after~opening } { #2 }
2664 \@@_rec_preamble:n
2665 }
2666 { \@@_rec_preamble:n #2 }
2667 }

```

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

```

2668 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2669 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2670 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2671 {
2672 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2673 \tl_if_in:nnTF { ) ] \} } { #2 }
2674 { \@@_make_preamble_v:nnn #1 #2 }
2675 {
2676 \str_if_eq:nnTF { \s_stop } { #2 }
2677 {
2678 \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2679 { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2680 {
2681 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2682         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2683         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2684         \@@_rec_preamble:n #2
2685     }
2686 }
2687 {
2688     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2689     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2690     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2691     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2692     \@@_rec_preamble:n #2
2693 }
2694 }
2695 }
2696 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2697 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }

2698 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2699 {
2700     \str_if_eq:nnTF { \s_stop } { #3 }
2701     {
2702         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2703         {
2704             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2705             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2706             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2707             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2708         }
2709         {
2710             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2711             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2712             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2713             \@@_error:nn { double~closing~delimiter } { #2 }
2714         }
2715     }
2716     {
2717         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2718         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2719         \@@_error:nn { double~closing~delimiter } { #2 }
2720         \@@_rec_preamble:n #3
2721     }
2722 }

2723 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2724 { \use:c { @@ _ \token_to_str:N ) : } }

```

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

```

2725 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2726 {
2727     \str_if_eq:nnTF { #1 } { < }
2728     { \@@_rec_preamble_after_col_i:n }
2729     {
2730         \str_if_eq:nnTF { #1 } { @ }
2731         { \@@_rec_preamble_after_col_ii:n }
2732         {
2733             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2734             {
2735                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2736                 { ! { \skip_horizontal:N \arrayrulewidth } }
2737             }
2738         }

```



```

2739         \clist_if_in:NeT \l_@@_vlines_clist
2740         { \int_eval:n { \c@jCol + 1 } }
2741         {
2742             \tl_gput_right:Nn \g_@@_array_preamble_tl
2743             { ! { \skip_horizontal:N \arrayrulewidth } }
2744         }
2745     }
2746     \@@_rec_preamble:n { #1 }
2747 }
2748 }
2749 }
2750 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2751 {
2752     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2753     \@@_rec_preamble_after_col:n
2754 }

```

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

```

2755 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2756 {
2757     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2758     {
2759         \tl_gput_right:Nn \g_@@_array_preamble_tl
2760         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2761     }
2762     {
2763         \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2764         {
2765             \tl_gput_right:Nn \g_@@_array_preamble_tl
2766             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2767         }
2768         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2769     }
2770     \@@_rec_preamble:n
2771 }
2772 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2773 {
2774     \tl_clear:N \l_tmpa_tl
2775     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2776     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2777 }

```

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.

```

2778 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2779 { \@@_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.

```

2780 \cs_new_protected:Npn \@@_X: #1 #2
2781 {
2782     \str_if_eq:nnTF { #2 } { [ ]
2783         { \@@_make_preamble_X:w [ ] }
2784         { \@@_make_preamble_X:w [ ] #2 }
2785     }
2786 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2787 { \@@_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 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`.

```

2788 \keys_define:nn { nicematrix / X-column }
2789 {
2790   unknown .code:n =
2791     \regex_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2792     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2793     { \@@_error_or_warning:n { invalid-weight } }
2794 }

```

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

```

2795 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2796 {

```

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).

```

2797   \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).

```

2798   \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`).

```

2799   \fp_set:Nn \l_tmpa_fp { 1.0 }
2800   \@@_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 now in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitly provided by the final user.

```

2801   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2802   \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).

```

2803   \bool_if:NTF \l_@@_X_columns_aux_bool
2804   {
2805     \@@_make_preamble_ii_iv:nnn

```

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

```

2806     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2807     { minipage }
2808     { \@@_no_update_width: }
2809   }

```

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.

```

2810   {
2811     \tl_gput_right:Nn \g_@@_array_preamble_tl
2812     {
2813       > {
2814         \@@_cell_begin:
2815         \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`.

```

2816   \NotEmpty

```

The following code will nullify the box of the cell.

```
2817         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2818         { \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`.

```
2819         \begin { minipage } { 5 cm } \arraybackslash
2820     }
2821     c
2822     < {
2823         \end { minipage }
2824         \@@_cell_end:
2825     }
2826 }
2827 \int_gincr:N \c@jCol
2828 \@@_rec_preamble_after_col:n
2829 }
2830 }

2831 \cs_new_protected:Npn \@@_no_update_width:
2832 {
2833     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2834     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2835 }
```

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

```
2836 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2837 {
2838     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2839     { \int_eval:n { \c@jCol + 1 } }
2840     \tl_gput_right:Ne \g_@@_array_preamble_tl
2841     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2842     \@@_rec_preamble:n
2843 }
```

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.

```
2844 \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 forgotten the preamble of its environment).

```
2845 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2846 { \@@_fatal:n { Preamble-forgotten } }
2847 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2848 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2849 { @@ _ \token_to_str:N \hline : }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2851 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2852 { @@ _ \token_to_str:N \hline : }
2853 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2854 { @@ _ \token_to_str:N \hline : }
2855 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2856 { @@ _ \token_to_str:N \hline : }
```

## 12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```
2857 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2858 {
```

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`.

```

2859 \multispan { #1 }
2860 \cs_set_eq:NN \@_update_max_cell_width: \prg_do_nothing:
2861 \begingroup
2862 \bool_if:NT \c_@@_testphase_table_bool
2863 { \tbl_update_multicolumn_cell_data:n { #1 } }
2864 \def \@addamp
2865 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2866 \tl_gclear:N \g_@@_preamble_tl
2867 \@_make_m_preamble:n #2 \q_stop

```

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

```

2868 \exp_args:No \mkpream \g_@@_preamble_tl
2869 \@addtopreamble \@empty
2870 \endgroup
2871 \bool_if:NT \c_@@_recent_array_bool
2872 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2873 \int_compare:nNt { #1 } > { \c_one_int }
2874 {
2875   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2876   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2877   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2878   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2879   {
2880     {
2881       \int_if_zero:nTF { \c@jCol }
2882       { \int_eval:n { \c@iRow + 1 } }
2883       { \int_use:N \c@iRow }
2884     }
2885     { \int_eval:n { \c@jCol + 1 } }
2886     {
2887       \int_if_zero:nTF { \c@jCol }
2888       { \int_eval:n { \c@iRow + 1 } }
2889       { \int_use:N \c@iRow }
2890     }
2891     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2892 { }
2893 }
2894 }

```

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

```

2895 \RenewDocumentCommand \cellcolor { 0 { } m }
2896 {
2897   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2898   {
2899     \@_rectanglecolor [ ##1 ]
2900     { \exp_not:n { ##2 } }
2901     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2902     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2903   }
2904   \ignorespaces
2905 }

```

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

```
2906 \def \@sharp { #3 }
2907 \@arstrut
2908 \@preamble
2909 \null
```

We add some lines.

```
2910 \int_gadd:Nn \c@jCol { #1 - 1 }
2911 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
2912 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2913 \ignorespaces
2914 }
```

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`.

```
2915 \cs_new_protected:Npn \@_make_m_preamble:n #1
2916 {
2917   \str_case:nnF { #1 }
2918   {
2919     c { \@_make_m_preamble_i:n #1 }
2920     l { \@_make_m_preamble_i:n #1 }
2921     r { \@_make_m_preamble_i:n #1 }
2922     > { \@_make_m_preamble_ii:nn #1 }
2923     ! { \@_make_m_preamble_ii:nn #1 }
2924     @ { \@_make_m_preamble_ii:nn #1 }
2925     | { \@_make_m_preamble_iii:n #1 }
2926     p { \@_make_m_preamble_iv:nnn t #1 }
2927     m { \@_make_m_preamble_iv:nnn c #1 }
2928     b { \@_make_m_preamble_iv:nnn b #1 }
2929     w { \@_make_m_preamble_v:nnnn { } #1 }
2930     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2931     \q_stop { }
2932   }
2933   {
2934     \cs_if_exist:cTF { NC @ find @ #1 }
2935     {
2936       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2937       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2938     }
2939     {
2940       \str_if_eq:nnTF { #1 } { S }
2941       { \@_fatal:n { unknown~column~type~S } }
2942       { \@_fatal:nn { unknown~column~type } { #1 } }
2943     }
2944   }
2945 }
```

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

```
2946 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2947 {
2948   \tl_gput_right:Nn \g_@@_preamble_tl
2949   {
2950     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2951     #1
2952     < \@_cell_end:
2953   }
```

We test for the presence of a `<`.

```
2954 \@_make_m_preamble_x:n
2955 }
```

For >, ! and @

```

2956 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2957 {
2958   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2959   \@@_make_m_preamble:n
2960 }

```

For |

```

2961 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2962 {
2963   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2964   \@@_make_m_preamble:n
2965 }

```

For p, m and b

```

2966 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2967 {
2968   \tl_gput_right:Nn \g_@@_preamble_tl
2969   {
2970     > {
2971       \@@_cell_begin:
2972       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2973       \mode_leave_vertical:
2974       \arraybackslash
2975       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2976     }
2977     c
2978     < {
2979       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2980       \end { minipage }
2981       \@@_cell_end:
2982     }
2983   }

```

We test for the presence of a <.

```

2984   \@@_make_m_preamble_x:n
2985 }

```

For w and W

```

2986 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2987 {
2988   \tl_gput_right:Nn \g_@@_preamble_tl
2989   {
2990     > {
2991       \dim_set:Nn \l_@@_col_width_dim { #4 }
2992       \hbox_set:Nw \l_@@_cell_box
2993       \@@_cell_begin:
2994       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2995     }
2996     c
2997     < {
2998       \@@_cell_end:
2999       \hbox_set_end:
3000       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3001       #1
3002       \@@_adjust_size_box:
3003       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3004     }
3005   }

```

We test for the presence of a <.

```

3006   \@@_make_m_preamble_x:n
3007 }

```

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

```

3008 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3009 {
3010   \str_if_eq:nnTF { #1 } { < }
3011     { \@@_make_m_preamble_ix:n }
3012     { \@@_make_m_preamble:n { #1 } }
3013 }
3014 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3015 {
3016   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3017   \@@_make_m_preamble_x:n
3018 }

```

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`).

```

3019 \cs_new_protected:Npn \@@_put_box_in_flow:
3020 {
3021   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3022   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3023   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3024     { \box_use_drop:N \l_tmpa_box }
3025     { \@@_put_box_in_flow_i: }
3026 }

```

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

```

3027 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3028 {
3029   \pgfpicture
3030     \@@_qpoint:n { row - 1 }
3031     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3032     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3033     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3034     \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).

```

3035   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3036   {
3037     \int_set:Nn \l_tmpa_int
3038     {
3039       \str_range:Nnn
3040         \l_@@_baseline_tl
3041         6
3042         { \tl_count:o \l_@@_baseline_tl }
3043     }
3044     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3045   }
3046   {
3047     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3048     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3049     {
3050       \str_if_eq:onTF \l_@@_baseline_tl { b }
3051       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3052       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3053     }
3054     \bool_lazy_or:nnT
3055       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3056       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }

```

```

3057         {
3058             \@@_error:n { bad-value-for-baseline }
3059             \int_set_eq:NN \l_tmpa_int \c_one_int
3060         }
3061         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3062         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3063     }
3064     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3065     \endpgfpicture
3066     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3067     \box_use_drop:N \l_tmpa_box
3068 }

```

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).

```

3069 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3070 {

```

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.

```

3071     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3072     {
3073         \int_compare:nNnT { \c_jCol } > { \c_one_int }
3074         {
3075             \box_set_wd:Nn \l_@@_the_array_box
3076             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3077         }
3078     }

```

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

```

3079     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3080     \bool_if:NT \l_@@_caption_above_bool
3081     {
3082         \tl_if_empty:NF \l_@@_caption_tl
3083         {
3084             \bool_set_false:N \g_@@_caption_finished_bool
3085             \int_gzero:N \c@tabularnote
3086             \@@_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).

```

3087         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3088         {
3089             \tl_gput_right:Ne \g_@@_aux_tl
3090             {
3091                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3092                 { \int_use:N \g_@@_notes_caption_int }
3093             }
3094             \int_gzero:N \g_@@_notes_caption_int
3095         }
3096     }
3097 }

```



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

```

3098   \hbox
3099   {
3100   \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right now 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.

```

3101   \@@_create_extra_nodes:
3102   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3103 }

```

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).

```

3104   \bool_lazy_any:nT
3105   {
3106     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3107     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3108     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3109   }
3110   \@@_insert_tabularnotes:
3111   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3112   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3113   \end { minipage }
3114 }

```

```

3115 \cs_new_protected:Npn \@@_insert_caption:
3116 {
3117   \tl_if_empty:NF \l_@@_caption_tl
3118   {
3119     \cs_if_exist:NTF \capttype
3120     { \@@_insert_caption_i: }
3121     { \@@_error:n { caption~outside~float } }
3122   }
3123 }

```

```

3124 \cs_new_protected:Npn \@@_insert_caption_i:
3125 {
3126   \group_begin:

```

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

```

3127   \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.

```

3128   \IfPackageLoadedT { floatrow }
3129   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3130   \tl_if_empty:NTF \l_@@_short_caption_tl
3131   { \caption }
3132   { \caption [ \l_@@_short_caption_tl ] }
3133   { \l_@@_caption_tl }

```

In some circonstancies (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.

```

3134 \bool_if:NF \g_@@_caption_finished_bool
3135 {
3136   \bool_gset_true:N \g_@@_caption_finished_bool
3137   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3138   \int_gzero:N \c@tabularnote
3139 }
3140 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3141 \group_end:
3142 }
3143 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3144 {
3145   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3146   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3147 }
3148 \cs_new_protected:Npn \@@_insert_tabularnotes:
3149 {
3150   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3151   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3152   \skip_vertical:N 0.65ex

```

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

```

3153 \group_begin:
3154 \l_@@_notes_code_before_tl
3155 \tl_if_empty:NF \g_@@_tabularnote_tl
3156 {
3157   \g_@@_tabularnote_tl \par
3158   \tl_gclear:N \g_@@_tabularnote_tl
3159 }

```

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.

```

3160 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3161 {
3162   \bool_if:NTF \l_@@_notes_para_bool
3163   {
3164     \begin { tabularnotes* }
3165     \seq_map_inline:Nn \g_@@_notes_seq
3166       { \@@_one_tabularnote:nn ##1 }
3167     \strut
3168     \end { tabularnotes* }

```

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

```

3169 \par
3170 }
3171 {
3172   \tabularnotes
3173   \seq_map_inline:Nn \g_@@_notes_seq
3174     { \@@_one_tabularnote:nn ##1 }
3175   \strut
3176   \endtabularnotes
3177 }
3178 }
3179 \unskip
3180 \group_end:
3181 \bool_if:NT \l_@@_notes_bottomrule_bool
3182 {
3183   \IfPackageLoadedTF { booktabs }
3184   {

```

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

```

3185 \skip_vertical:N \aboverulesep

```

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

```

3186         { \CT@arc@ \hrule height \heavyrulewidth }
3187     }
3188     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3189 }
3190 \l_@@_notes_code_after_tl
3191 \seq_gclear:N \g_@@_notes_seq
3192 \seq_gclear:N \g_@@_notes_in_caption_seq
3193 \int_gzero:N \c@tabularnote
3194 }

```

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.

```

3195 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3196 {
3197     \tl_if_novalue:nTF { #1 }
3198     { \item }
3199     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3200 }

```

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.

```

3201 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3202 {
3203     \pgfpicture
3204     \@@_qpoint:n { row - 1 }
3205     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3206     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3207     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3208     \endpgfpicture
3209     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3210     \int_if_zero:nT { \l_@@_first_row_int }
3211     {
3212         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3213         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3214     }
3215     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3216 }

```

Now, the general case.

```

3217 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3218 {

```

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

```

3219     \str_if_eq:eeT \l_@@_baseline_tl { t }
3220     { \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`.

```

3221     \pgfpicture
3222     \@@_qpoint:n { row - 1 }
3223     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3224     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3225     {
3226         \int_set:Nn \l_tmpa_int
3227         {
3228             \str_range:Nnn
3229             \l_@@_baseline_tl
3230             { 6 }
3231             { \tl_count:o \l_@@_baseline_tl }

```

```

3232     }
3233     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3234 }
3235 {
3236     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3237     \bool_lazy_or:nnT
3238         { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3239         { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3240     {
3241         \@@_error:n { bad-value-for~baseline }
3242         \int_set:Nn \l_tmpa_int 1
3243     }
3244     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3245 }
3246 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3247 \endpgfpicture
3248 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3249 \int_if_zero:nT { \l_@@_first_row_int }
3250 {
3251     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3252     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3253 }
3254 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3255 }

```

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.

```

3256 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3257 {

```

We will compute the real width of both delimiters used.

```

3258     \dim_zero_new:N \l_@@_real_left_delim_dim
3259     \dim_zero_new:N \l_@@_real_right_delim_dim
3260     \hbox_set:Nn \l_tmpb_box
3261     {
3262         \m@th % added 2024/11/21
3263         \c_math_toggle_token
3264         \left #1
3265         \vcenter
3266         {
3267             \vbox_to_ht:nn
3268                 { \box_ht_plus_dp:N \l_tmpa_box }
3269                 { }
3270         }
3271         \right .
3272         \c_math_toggle_token
3273     }
3274     \dim_set:Nn \l_@@_real_left_delim_dim
3275         { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3276     \hbox_set:Nn \l_tmpb_box
3277     {
3278         \m@th % added 2024/11/21
3279         \c_math_toggle_token
3280         \left .
3281         \vbox_to_ht:nn
3282             { \box_ht_plus_dp:N \l_tmpa_box }
3283             { }
3284         \right #2
3285         \c_math_toggle_token
3286     }
3287     \dim_set:Nn \l_@@_real_right_delim_dim
3288         { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3289 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3290 \@@_put_box_in_flow:
3291 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3292 }

```

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).

```

3293 \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).

```

3294 {
3295   \peek_remove_spaces:n
3296   {
3297     \peek_meaning:NTF \end
3298     { \@@_analyze_end:Nn }
3299     {
3300       \@@_transform_preamble:

```

Here is the call to `\array` (we have a dedicated macro `\@@_array:n` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3301       \@@_array:o \g_@@_array_preamble_tl
3302     }
3303   }
3304 }
3305 {
3306   \@@_create_col_nodes:
3307   \endarray
3308 }

```

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

```

3309 \NewDocumentEnvironment { @@-light-syntax } { b }
3310 {

```

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`.

```

3311 \tl_if_empty:nT { #1 }
3312 { \@@_fatal:n { empty-environment } }
3313 \tl_if_in:nnT { #1 } { & }
3314 { \@@_fatal:n { ampersand-in-light-syntax } }
3315 \tl_if_in:nnT { #1 } { \ }
3316 { \@@_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`.

```

3317 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3318 }

```

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.

```

3319 {
3320   \@@_create_col_nodes:
3321   \endarray
3322 }

3323 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3324 {
3325   \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).

```

3326   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3327   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3328   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3329     { \seq_set_split:Nee }
3330     { \seq_set_split:Non }
3331   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3332   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3333   \tl_if_empty:NF \l_tmpa_tl
3334   { \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.

```

3335   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3336     { \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`).

```

3337   \tl_build_begin:N \l_@@_new_body_tl
3338   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3339   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3340   \@@_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).

```

3341   \seq_map_inline:Nn \l_@@_rows_seq
3342   {
3343     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3344     \@@_line_with_light_syntax:n { ##1 }
3345   }
3346   \tl_build_end:N \l_@@_new_body_tl

3347   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3348   {
3349     \int_set:Nn \l_@@_last_col_int
3350       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3351   }

```

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.

```

3352   \@@_transform_preamble:

```

The call to `\array` is in the following command (we have a dedicated macro `\@@_array:` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3353   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3354 }

```

```

3355 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3356 {
3357   \seq_clear_new:N \l_@@_cells_seq
3358   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3359   \int_set:Nn \l_@@_nb_cols_int
3360   {
3361     \int_max:nn
3362     { \l_@@_nb_cols_int }
3363     { \seq_count:N \l_@@_cells_seq }
3364   }
3365   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3366   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3367   \seq_map_inline:Nn \l_@@_cells_seq
3368   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3369 }
3370 \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`.

```

3371 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3372 {
3373   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3374   { \@@_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.

```

3375   \end { #2 }
3376 }

```

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`).

```

3377 \cs_new:Npn \@@_create_col_nodes:
3378 {
3379   \crrr
3380   \int_if_zero:nT { \l_@@_first_col_int }
3381   {
3382     \omit
3383     \hbox_overlap_left:n
3384     {
3385       \bool_if:NT \l_@@_code_before_bool
3386       { \pgfsys@markposition { \@@_env: - col - 0 } }
3387       \pgfpicture
3388       \pgfrememberpicturepositiononpagetrue
3389       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3390       \str_if_empty:NF \l_@@_name_str
3391       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3392       \endpgfpicture
3393       \skip_horizontal:n { 2 \colsep + \g_@@_width_first_col_dim }
3394     }
3395     &
3396   }
3397   \omit

```

The following instruction must be put after the instruction `\omit`.

```

3398   \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`).

```

3399   \int_if_zero:nTF { \l_@@_first_col_int }
3400   {
3401     \bool_if:NT \l_@@_code_before_bool

```

```

3402     {
3403         \hbox
3404         {
3405             \skip_horizontal:n { -0.5 \arrayrulewidth }
3406             \pgfsys@markposition { \@@_env: - col - 1 }
3407             \skip_horizontal:n { 0.5 \arrayrulewidth }
3408         }
3409     }
3410     \pgfpicture
3411     \pgfrememberpicturepositiononpagetrue
3412     \pgfcoordinate { \@@_env: - col - 1 }
3413     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3414     \str_if_empty:NF \l_@@_name_str
3415     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3416     \endpgfpicture
3417 }
3418 {
3419     \bool_if:NT \l_@@_code_before_bool
3420     {
3421         \hbox
3422         {
3423             \skip_horizontal:n { 0.5 \arrayrulewidth }
3424             \pgfsys@markposition { \@@_env: - col - 1 }
3425             \skip_horizontal:n { -0.5 \arrayrulewidth }
3426         }
3427     }
3428     \pgfpicture
3429     \pgfrememberpicturepositiononpagetrue
3430     \pgfcoordinate { \@@_env: - col - 1 }
3431     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3432     \str_if_empty:NF \l_@@_name_str
3433     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3434     \endpgfpicture
3435 }

```

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.

```

3436     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3437     \bool_if:NF \l_@@_auto_columns_width_bool
3438     { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3439     {
3440         \bool_lazy_and:nnTF
3441         { \l_@@_auto_columns_width_bool }
3442         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3443         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3444         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3445         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3446     }
3447     \skip_horizontal:N \g_tmpa_skip
3448     \hbox
3449     {
3450         \bool_if:NT \l_@@_code_before_bool
3451         {
3452             \hbox
3453             {
3454                 \skip_horizontal:n { -0.5 \arrayrulewidth }
3455                 \pgfsys@markposition { \@@_env: - col - 2 }
3456                 \skip_horizontal:n { 0.5 \arrayrulewidth }
3457             }

```



```

3458     }
3459     \pgfpicture
3460     \pgfrememberpicturepositiononpagetrue
3461     \pgfcoordinate { \@@_env: - col - 2 }
3462     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3463     \str_if_empty:NF \l_@@_name_str
3464     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3465     \endpgfpicture
3466 }

```

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.

```

3467     \int_gset_eq:NN \g_tmpa_int \c_one_int
3468     \bool_if:NTF \g_@@_last_col_found_bool
3469     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3470     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3471     {
3472     &
3473     \omit
3474     \int_gincr:N \g_tmpa_int

```

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

```

3475     \skip_horizontal:N \g_tmpa_skip
3476     \bool_if:NT \l_@@_code_before_bool
3477     {
3478     \hbox
3479     {
3480     \skip_horizontal:n { -0.5 \arrayrulewidth }
3481     \pgfsys@markposition
3482     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3483     \skip_horizontal:n { 0.5 \arrayrulewidth }
3484     }
3485     }

```

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

```

3486     \pgfpicture
3487     \pgfrememberpicturepositiononpagetrue
3488     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3490     \str_if_empty:NF \l_@@_name_str
3491     {
3492     \pgfnodealias
3493     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3494     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3495     }
3496     \endpgfpicture
3497 }

3498 &
3499 \omit

```

The two following lines have been added on 2021-12-15 to solve a bug mentioned by Joao Luis Soares by mail.

```

3500     \int_if_zero:nT { \g_@@_col_total_int }
3501     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3502     \skip_horizontal:N \g_tmpa_skip
3503     \int_gincr:N \g_tmpa_int
3504     \bool_lazy_any:nF
3505     {
3506     \g_@@_delims_bool
3507     \l_@@_tabular_bool
3508     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3509     \l_@@_exterior_arraycolsep_bool
3510     \l_@@_bar_at_end_of_pream_bool

```

```

3511     }
3512     { \skip_horizontal:n { - \col@sep } }
3513 \bool_if:NT \l_@@_code_before_bool
3514 {
3515     \hbox
3516     {
3517         \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.

```

3518         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3519         { \skip_horizontal:n { - \arraycolsep } }
3520         \pgfsys@markposition
3521         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3522         \skip_horizontal:n { 0.5 \arrayrulewidth }
3523         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3524         { \skip_horizontal:N \arraycolsep }
3525     }
3526 }
3527 \pgfpicture
3528 \pgfrememberpicturepositiononpagetrue
3529 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3530 {
3531     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3532     {
3533         \pgfpoint
3534         { - 0.5 \arrayrulewidth - \arraycolsep }
3535         \c_zero_dim
3536     }
3537     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3538 }
3539 \str_if_empty:NF \l_@@_name_str
3540 {
3541     \pgfnodealias
3542     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3543     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3544 }
3545 \endpgfpicture

3546 \bool_if:NT \g_@@_last_col_found_bool
3547 {
3548     \hbox_overlap_right:n
3549     {
3550         \skip_horizontal:N \g_@@_width_last_col_dim
3551         \skip_horizontal:N \col@sep
3552         \bool_if:NT \l_@@_code_before_bool
3553         {
3554             \pgfsys@markposition
3555             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3556         }
3557         \pgfpicture
3558         \pgfrememberpicturepositiononpagetrue
3559         \pgfcoordinate
3560         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3561         \pgfpointorigin
3562         \str_if_empty:NF \l_@@_name_str
3563         {
3564             \pgfnodealias
3565             {
3566                 \l_@@_name_str - col
3567                 - \int_eval:n { \g_@@_col_total_int + 1 }
3568             }

```

```

3569         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3570     }
3571     \endpgfpicture
3572 }
3573 }
3574 % \cr
3575 }

```

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

```

3576 \tl_const:Nn \c_@@_preamble_first_col_tl
3577 {
3578     >
3579     {

```

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

```

3580         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3581         \bool_gset_true:N \g_@@_after_col_zero_bool
3582         \@@_begin_of_row:
3583         \hbox_set:Nw \l_@@_cell_box
3584         \@@_math_toggle:
3585         \@@_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”.

```

3586         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3587         {
3588             \bool_lazy_or:nnT
3589             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3590             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3591             {
3592                 \l_@@_code_for_first_col_tl
3593                 \xglobal \colorlet { nicematrix-first-col } { . }
3594             }
3595         }
3596     }

```

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`.

```

3597     l
3598     <
3599     {
3600         \@@_math_toggle:
3601         \hbox_set_end:
3602         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3603         \@@_adjust_size_box:
3604         \@@_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.

```

3605         \dim_gset:Nn \g_@@_width_first_col_dim
3606         { \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.

```

3607     \hbox_overlap_left:n
3608     {
3609         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3610         { \@@_node_cell: }
3611         { \box_use_drop:N \l_@@_cell_box }
3612         \skip_horizontal:N \l_@@_left_delim_dim
3613         \skip_horizontal:N \l_@@_left_margin_dim
3614         \skip_horizontal:N \l_@@_extra_left_margin_dim
3615     }
3616     \bool_gset_false:N \g_@@_empty_cell_bool

```

```

3617     \skip_horizontal:n { -2 \col@sep }
3618   }
3619 }

```

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

```

3620 \tl_const:Nn \c_@@_preamble_last_col_tl
3621 {
3622   >
3623   {
3624     \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).

```

3625     \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.

```

3626     \bool_gset_true:N \g_@@_last_col_found_bool
3627     \int_gincr:N \c@jCol
3628     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3629     \hbox_set:Nw \l_@@_cell_box
3630     \@@_math_toggle:
3631     \@@_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”.

```

3632     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3633     {
3634       \bool_lazy_or:nnT
3635       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3636       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3637       {
3638         \l_@@_code_for_last_col_tl
3639         \xglobal \colorlet { nicematrix-last-col } { . }
3640       }
3641     }
3642   }
3643   1
3644   <
3645   {
3646     \@@_math_toggle:
3647     \hbox_set_end:
3648     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3649     \@@_adjust_size_box:
3650     \@@_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.

```

3651     \dim_gset:Nn \g_@@_width_last_col_dim
3652     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3653     \skip_horizontal:n { -2 \col@sep }

```

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

```

3654     \hbox_overlap_right:n
3655     {
3656       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3657       {
3658         \skip_horizontal:N \l_@@_right_delim_dim
3659         \skip_horizontal:N \l_@@_right_margin_dim
3660         \skip_horizontal:N \l_@@_extra_right_margin_dim
3661         \@@_node_cell:
3662       }
3663     }
3664     \bool_gset_false:N \g_@@_empty_cell_bool
3665   }
3666 }

```

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

```

3667 \NewDocumentEnvironment { NiceArray } { }
3668 {
3669   \bool_gset_false:N \g_@@_delims_bool
3670   \str_if_empty:NT \g_@@_name_env_str
3671   { \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).

```

3672   \NiceArrayWithDelims . .
3673 }
3674 { \endNiceArrayWithDelims }

```

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

```

3675 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3676 {
3677   \NewDocumentEnvironment { #1 NiceArray } { }
3678   {
3679     \bool_gset_true:N \g_@@_delims_bool
3680     \str_if_empty:NT \g_@@_name_env_str
3681     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3682     \@@_test_if_math_mode:
3683     \NiceArrayWithDelims #2 #3
3684   }
3685   { \endNiceArrayWithDelims }
3686 }
3687 \@@_def_env:NNN p ( )
3688 \@@_def_env:NNN b [ ]
3689 \@@_def_env:NNN B \{ \}
3690 \@@_def_env:NNN v \vert \vert
3691 \@@_def_env:NNN V \Vert \Vert

```

## 13 The environment `{NiceMatrix}` and its variants

```

3692 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3693 {
3694   \bool_set_false:N \l_@@_preamble_bool
3695   \tl_clear:N \l_tmpa_tl
3696   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3697   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3698   \tl_put_right:Nn \l_tmpa_tl
3699   {
3700     *
3701     {
3702       \int_case:nnF \l_@@_last_col_int
3703       {
3704         { -2 } { \c@MaxMatrixCols }
3705         { -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).

```

3706       }
3707       { \int_eval:n { \l_@@_last_col_int - 1 } }
3708     }
3709     { #2 }
3710   }
3711   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3712   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3713 }

```

```

3714 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3715 \clist_map_inline:nn { p , b , B , v , V }
3716 {
3717   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3718   {
3719     \bool_gset_true:N \g_@@_delims_bool
3720     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3721     \int_if_zero:nT { \l_@@_last_col_int }
3722     {
3723       \bool_set_true:N \l_@@_last_col_without_value_bool
3724       \int_set:Nn \l_@@_last_col_int { -1 }
3725     }
3726     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3727     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3728   }
3729   { \use:c { end #1 NiceArray } }
3730 }

```

We define also an environment {NiceMatrix}

```

3731 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3732 {
3733   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3734   \int_if_zero:nT { \l_@@_last_col_int }
3735   {
3736     \bool_set_true:N \l_@@_last_col_without_value_bool
3737     \int_set:Nn \l_@@_last_col_int { -1 }
3738   }
3739   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3740   \bool_lazy_or:nnT
3741   { \clist_if_empty_p:N \l_@@_vlines_clist }
3742   { \l_@@_except_borders_bool }
3743   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3744   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3745 }
3746 { \endNiceArray }

```

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

```

3747 \cs_new_protected:Npn \@@_NotEmpty:
3748 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3749 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3750 {

```

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.

```

3751   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3752   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3753   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3754   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3755   \tl_if_empty:NF \l_@@_short_caption_tl
3756   {
3757     \tl_if_empty:NT \l_@@_caption_tl
3758     {
3759       \@@_error_or_warning:n { short-caption-without~caption }
3760       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3761     }
3762   }
3763   \tl_if_empty:NF \l_@@_label_tl
3764   {

```

```

3765     \tl_if_empty:NT \l_@@_caption_tl
3766     { \@@_error_or_warning:n { label~without~caption } }
3767   }
3768   \NewDocumentEnvironment { TabularNote } { b }
3769   {
3770     \bool_if:NTF \l_@@_in_code_after_bool
3771     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3772     {
3773       \tl_if_empty:NF \g_@@_tabularnote_tl
3774       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3775       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3776     }
3777   }
3778   { }
3779   \@@_settings_for_tabular:
3780   \NiceArray { #2 }
3781 }
3782 { \endNiceArray }
3783 \cs_new_protected:Npn \@@_settings_for_tabular:
3784 {
3785   \bool_set_true:N \l_@@_tabular_bool
3786   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3787   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3788   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3789 }

3790 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3791 {
3792   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3793   \dim_set:Nn \l_@@_width_dim { #1 }
3794   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3795   \@@_settings_for_tabular:
3796   \NiceArray { #3 }
3797 }
3798 {
3799   \endNiceArray
3800   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3801   { \@@_error:n { NiceTabularX~without~X } }
3802 }

3803 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3804 {
3805   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3806   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3807   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3808   \@@_settings_for_tabular:
3809   \NiceArray { #3 }
3810 }
3811 { \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`).

```

3812 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3813 {
3814   \bool_lazy_all:nT
3815   {

```

```

3816 { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3817 { \l_@@_hvlines_bool }
3818 { ! \g_@@_delims_bool }
3819 { ! \l_@@_except_borders_bool }
3820 }
3821 {
3822   \bool_set_true:N \l_@@_except_borders_bool
3823   \clist_if_empty:NF \l_@@_corners_clist
3824   { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3825   \tl_gput_right:Nn \g_@@_pre_code_after_tl
3826   {
3827     \@@_stroke_block:nnn
3828     {
3829       rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3830       draw = \l_@@_rules_color_tl
3831     }
3832     { 1-1 }
3833     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3834   }
3835 }
3836 }

```

```

3837 \cs_new_protected:Npn \@@_after_array:
3838 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` 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`.

```

3839   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3840   \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.

```

3841   \bool_if:NT \g_@@_last_col_found_bool
3842   { \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`.

```

3843   \bool_if:NT \l_@@_last_col_without_value_bool
3844   { \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.

```

3845   \bool_if:NT \l_@@_last_row_without_value_bool
3846   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3847   \tl_gput_right:Ne \g_@@_aux_tl
3848   {
3849     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3850     {
3851       \int_use:N \l_@@_first_row_int ,
3852       \int_use:N \c@iRow ,
3853       \int_use:N \g_@@_row_total_int ,
3854       \int_use:N \l_@@_first_col_int ,
3855       \int_use:N \c@jCol ,
3856       \int_use:N \g_@@_col_total_int
3857     }
3858   }

```



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`).

```

3859 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3860 {
3861   \tl_gput_right:Ne \g_@@_aux_tl
3862   {
3863     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3864     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3865   }
3866 }
3867 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3868 {
3869   \tl_gput_right:Ne \g_@@_aux_tl
3870   {
3871     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3872     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3873     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3874     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3875   }
3876 }

```

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

```

3877 \@@_create_diag_nodes:

```

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

```

3878 \pgfpicture
3879 \@@_create_aliases_last:
3880 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3881 \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.

```

3882 \bool_if:NT \l_@@_parallelize_diags_bool
3883 {
3884   \int_gzero:N \g_@@_ddots_int
3885   \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.

```

3886 \dim_gzero:N \g_@@_delta_x_one_dim
3887 \dim_gzero:N \g_@@_delta_y_one_dim
3888 \dim_gzero:N \g_@@_delta_x_two_dim
3889 \dim_gzero:N \g_@@_delta_y_two_dim
3890 }
3891 \bool_set_false:N \l_@@_initial_open_bool
3892 \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.

```

3893 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3894 \@@_draw_dotted_lines:

```

---

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

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

```

3895   \clist_if_empty:NF \l_@@_corners_cells_clist
3896   {
3897     \bool_if:NTF \l_@@_no_cell_nodes_bool
3898     { \@@_error:n { corners~with~no~cell~nodes } }
3899     { \@@_compute_corners: }
3900   }

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3901   \@@_adjust_pos_of_blocks_seq:
3902   \@@_deal_with_rounded_corners:
3903   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3904   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3905   \IfPackageLoadedT { tikz }
3906   {
3907     \tikzset
3908     {
3909       every-picture / .style =
3910       {
3911         overlay ,
3912         remember~picture ,
3913         name~prefix = \@@_env: -
3914       }
3915     }
3916   }
3917   \bool_if:NT \c_@@_recent_array_bool
3918   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3919   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3920   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3921   \cs_set_eq:NN \OverBrace \@@_OverBrace
3922   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3923   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3924   \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.

```

3925   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3926   \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.

```

3927   \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).

```

3928   \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`).

```

3929   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3930   { \@@_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:`.

```

3931 \bool_set_true:N \l_@@_in_code_after_bool
3932 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3933 \scan_stop:
3934 \tl_gclear:N \g_nicematrix_code_after_tl
3935 \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.

```

3936 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3937 \tl_if_empty:NF \g_@@_pre_code_before_tl
3938 {
3939   \tl_gput_right:Ne \g_@@_aux_tl
3940   {
3941     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3942     { \exp_not:o \g_@@_pre_code_before_tl }
3943   }
3944   \tl_gclear:N \g_@@_pre_code_before_tl
3945 }
3946 \tl_if_empty:NF \g_nicematrix_code_before_tl
3947 {
3948   \tl_gput_right:Ne \g_@@_aux_tl
3949   {
3950     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3951     { \exp_not:o \g_nicematrix_code_before_tl }
3952   }
3953   \tl_gclear:N \g_nicematrix_code_before_tl
3954 }

3955 \str_gclear:N \g_@@_name_env_str
3956 \@@_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`.

```

3957 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3958 }

3959 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3960 {
3961   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3962   \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.

3963   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3964   { 0.6 \l_@@_xdots_shorten_start_dim }
3965   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3966   { 0.6 \l_@@_xdots_shorten_end_dim }
3967 }

```

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`.

```

3968 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3969 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

---

<sup>13</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

3970 \cs_new_protected:Npn \@@_create_alias_nodes:
3971 {
3972   \int_step_inline:nn { \c@iRow }
3973   {
3974     \pgfnodealias
3975     { \l_@@_name_str - ##1 - last }
3976     { \@@_env: - ##1 - \int_use:N \c@jCol }
3977   }
3978   \int_step_inline:nn { \c@jCol }
3979   {
3980     \pgfnodealias
3981     { \l_@@_name_str - last - ##1 }
3982     { \@@_env: - \int_use:N \c@iRow - ##1 }
3983   }
3984   \pgfnodealias % added 2025-04-05
3985   { \l_@@_name_str - last - last }
3986   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3987 }

```

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).

```

3988 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3989 {
3990   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3991   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3992 }

```

The following command must *not* be protected.

```

3993 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3994 {
3995   { #1 }
3996   { #2 }
3997   {
3998     \int_compare:nNnTF { #3 } > { 98 }
3999     { \int_use:N \c@iRow }
4000     { #3 }
4001   }
4002   {
4003     \int_compare:nNnTF { #4 } > { 98 }
4004     { \int_use:N \c@jCol }
4005     { #4 }
4006   }
4007   { #5 }
4008 }

```

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).

```

4009 \hook_gput_code:nnn { begindocument } { . }
4010 {
4011   \cs_new_protected:Npe \@@_draw_dotted_lines:
4012   {
4013     \c_@@_pgfortikzpicture_tl
4014     \@@_draw_dotted_lines_i:
4015     \c_@@_endpgfortikzpicture_tl
4016   }
4017 }

```

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

```

4018 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4019 {
4020   \pgfrememberpicturepositiononpagetrue
4021   \pgf@relevantforpicturesizefalse
4022   \g_@@_HVdotsfor_lines_tl
4023   \g_@@_Vdots_lines_tl
4024   \g_@@_Ddots_lines_tl
4025   \g_@@_Iddots_lines_tl
4026   \g_@@_Cdots_lines_tl
4027   \g_@@_Ldots_lines_tl
4028 }

4029 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4030 {
4031   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4032   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4033 }

```

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

```

4034 \pgfdeclareshape { @@_diag_node }
4035 {
4036   \savedanchor { \five }
4037   {
4038     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4039     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4040   }
4041   \anchor { 5 } { \five }
4042   \anchor { center } { \pgfpointorigin }
4043   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4044   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4045   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4046   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4047   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4048   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4049   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4050   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4051   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4052   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4053 }

```

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).

```

4054 \cs_new_protected:Npn \@@_create_diag_nodes:
4055 {
4056   \pgfpicture
4057   \pgfrememberpicturepositiononpagetrue
4058   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4059   {
4060     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4061     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4062     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4063     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4064     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4065     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4066     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4067     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4068     \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.

```

4069      \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4070      \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4071      \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4072      \str_if_empty:NF \l_@@_name_str
4073      { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4074    }

```

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

```

4075      \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4076      \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4077      \dim_set_eq:NN \l_tmpa_dim \pgf@y
4078      \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4079      \pgfcoordinate
4080      { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4081      \pgfnodealias
4082      { \@@_env: - last }
4083      { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4084      \str_if_empty:NF \l_@@_name_str
4085      {
4086        \pgfnodealias
4087        { \l_@@_name_str - \int_use:N \l_tmpa_int }
4088        { \@@_env: - \int_use:N \l_tmpa_int }
4089        \pgfnodealias
4090        { \l_@@_name_str - last }
4091        { \@@_env: - last }
4092      }
4093      \endpgfpicture
4094    }

```

## 16 We draw the dotted lines

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

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

The command `\@@_find_extremities_of_line: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.

```
4095 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4096 {
```

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

```
4097 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4098 \int_set:Nn \l_@@_initial_i_int { #1 }
4099 \int_set:Nn \l_@@_initial_j_int { #2 }
4100 \int_set:Nn \l_@@_final_i_int { #1 }
4101 \int_set:Nn \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.

```
4102 \bool_set_false:N \l_@@_stop_loop_bool
4103 \bool_do_until:Nn \l_@@_stop_loop_bool
4104 {
4105   \int_add:Nn \l_@@_final_i_int { #3 }
4106   \int_add:Nn \l_@@_final_j_int { #4 }
4107   \bool_set_false:N \l_@@_final_open_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.

```
4108   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4109     \if_int_compare:w #3 = \c_one_int
4110       \bool_set_true:N \l_@@_final_open_bool
4111     \else:
4112       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4113         \bool_set_true:N \l_@@_final_open_bool
4114       \fi:
4115     \fi:
4116   \else:
4117     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4118       \if_int_compare:w #4 = -1
4119         \bool_set_true:N \l_@@_final_open_bool
4120       \fi:
4121     \else:
4122       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4123         \if_int_compare:w #4 = \c_one_int
4124           \bool_set_true:N \l_@@_final_open_bool
4125         \fi:
4126       \fi:
4127     \fi:
4128   \fi:
4129   \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.

```
4130   {
```

We do a step backwards.

```
4131     \int_sub:Nn \l_@@_final_i_int { #3 }
4132     \int_sub:Nn \l_@@_final_j_int { #4 }
4133     \bool_set_true:N \l_@@_stop_loop_bool
4134   }
```

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`.

```
4135   {
4136     \cs_if_exist:cTF
4137     {
4138       @@ _ dotted _
```

```

4139         \int_use:N \l_@@_final_i_int -
4140         \int_use:N \l_@@_final_j_int
4141     }
4142     {
4143         \int_sub:Nn \l_@@_final_i_int { #3 }
4144         \int_sub:Nn \l_@@_final_j_int { #4 }
4145         \bool_set_true:N \l_@@_final_open_bool
4146         \bool_set_true:N \l_@@_stop_loop_bool
4147     }
4148     {
4149         \cs_if_exist:cTF
4150         {
4151             pgf @ sh @ ns @ \@@_env:
4152             - \int_use:N \l_@@_final_i_int
4153             - \int_use:N \l_@@_final_j_int
4154         }
4155         { \bool_set_true:N \l_@@_stop_loop_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.

```

4156         {
4157             \cs_set_nopar:cpn
4158             {
4159                 @@ _ dotted _
4160                 \int_use:N \l_@@_final_i_int -
4161                 \int_use:N \l_@@_final_j_int
4162             }
4163             { }
4164         }
4165     }
4166 }
4167 }

```

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

```

4168     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4169     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4170     \bool_do_until:Nn \l_@@_stop_loop_bool
4171     {
4172         \int_sub:Nn \l_@@_initial_i_int { #3 }
4173         \int_sub:Nn \l_@@_initial_j_int { #4 }
4174         \bool_set_false:N \l_@@_initial_open_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.

```

4175         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4176         \if_int_compare:w #3 = \c_one_int
4177         \bool_set_true:N \l_@@_initial_open_bool
4178         \else:

```

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

```

4179         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4180         \bool_set_true:N \l_@@_initial_open_bool
4181         \fi:
4182         \fi:
4183         \else:
4184         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4185         \if_int_compare:w #4 = \c_one_int

```



```

4186         \bool_set_true:N \l_@@_initial_open_bool
4187     \fi:
4188 \else:
4189     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4190         \if_int_compare:w #4 = -1
4191             \bool_set_true:N \l_@@_initial_open_bool
4192         \fi:
4193     \fi:
4194 \fi:
4195 \fi:
4196 \bool_if:NTF \l_@@_initial_open_bool
4197 {
4198     \int_add:Nn \l_@@_initial_i_int { #3 }
4199     \int_add:Nn \l_@@_initial_j_int { #4 }
4200     \bool_set_true:N \l_@@_stop_loop_bool
4201 }
4202 {
4203     \cs_if_exist:cTF
4204     {
4205         @@ _ dotted _
4206         \int_use:N \l_@@_initial_i_int -
4207         \int_use:N \l_@@_initial_j_int
4208     }
4209     {
4210         \int_add:Nn \l_@@_initial_i_int { #3 }
4211         \int_add:Nn \l_@@_initial_j_int { #4 }
4212         \bool_set_true:N \l_@@_initial_open_bool
4213         \bool_set_true:N \l_@@_stop_loop_bool
4214     }
4215     {
4216         \cs_if_exist:cTF
4217         {
4218             pgf @ sh @ ns @ \@@_env:
4219             - \int_use:N \l_@@_initial_i_int
4220             - \int_use:N \l_@@_initial_j_int
4221         }
4222         { \bool_set_true:N \l_@@_stop_loop_bool }
4223         {
4224             \cs_set_nopar:cpn
4225             {
4226                 @@ _ dotted _
4227                 \int_use:N \l_@@_initial_i_int -
4228                 \int_use:N \l_@@_initial_j_int
4229             }
4230             { }
4231         }
4232     }
4233 }
4234 }

```

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.

```

4235     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4236     {
4237         { \int_use:N \l_@@_initial_i_int }

```

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

```

4238         { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4239         { \int_use:N \l_@@_final_i_int }
4240         { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4241         { } % for the name of the block
4242     }
4243 }

```

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 analyse 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.

```

4244 \cs_new_protected:Npn \@@_open_shorten:
4245 {
4246   \bool_if:NT \l_@@_initial_open_bool
4247     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4248   \bool_if:NT \l_@@_final_open_bool
4249     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4250 }

```

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).

```

4251 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4252 {
4253   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4254   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4255   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4256   \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the `code-before`. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

4257   \seq_if_empty:NF \g_@@_submatrix_seq
4258   {
4259     \seq_map_inline:Nn \g_@@_submatrix_seq
4260       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4261   }
4262 }

```

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

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

```

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

```

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

```

4263 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4264 {
4265   \if_int_compare:w #3 > #1
4266   \else:
4267     \if_int_compare:w #1 > #5

```

```

4268 \else:
4269 \if_int_compare:w #4 > #2
4270 \else:
4271 \if_int_compare:w #2 > #6
4272 \else:
4273 \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4274 \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4275 \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4276 \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4277 \fi:
4278 \fi:
4279 \fi:
4280 \fi:
4281 }

4282 \cs_new_protected:Npn \@@_set_initial_coords:
4283 {
4284 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4285 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4286 }
4287 \cs_new_protected:Npn \@@_set_final_coords:
4288 {
4289 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4290 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4291 }
4292 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4293 {
4294 \pgfpointanchor
4295 {
4296 \@@_env:
4297 - \int_use:N \l_@@_initial_i_int
4298 - \int_use:N \l_@@_initial_j_int
4299 }
4300 { #1 }
4301 \@@_set_initial_coords:
4302 }
4303 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4304 {
4305 \pgfpointanchor
4306 {
4307 \@@_env:
4308 - \int_use:N \l_@@_final_i_int
4309 - \int_use:N \l_@@_final_j_int
4310 }
4311 { #1 }
4312 \@@_set_final_coords:
4313 }
4314 \cs_new_protected:Npn \@@_open_x_initial_dim:
4315 {
4316 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4317 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4318 {
4319 \cs_if_exist:cT
4320 { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4321 {
4322 \pgfpointanchor
4323 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4324 { west }
4325 \dim_set:Nn \l_@@_x_initial_dim
4326 { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4327 }
4328 }

```

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

```

4329 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4330 {
4331   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4332   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4333   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4334 }
4335 }
4336 \cs_new_protected:Npn \@@_open_x_final_dim:
4337 {
4338   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4339   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4340   {
4341     \cs_if_exist:cT
4342     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4343     {
4344       \pgfpointanchor
4345       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4346       { east }
4347       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4348       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4349     }
4350   }

```

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

```

4351 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4352 {
4353   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4354   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4355   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4356 }
4357 }

```

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.

```

4358 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4359 {
4360   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4361   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4362   {
4363     \@@_find_extremities_of_line: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.

```

4364 \group_begin:
4365 \@@_open_shorten:
4366 \int_if_zero:nTF { #1 }
4367 { \color { nicematrix-first-row } }
4368 {

```

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.

```

4369 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4370 { \color { nicematrix-last-row } }
4371 }
4372 \keys_set:nn { nicematrix / xdots } { #3 }
4373 \@@_color:o \l_@@_xdots_color_tl
4374 \@@_actually_draw_Ldots:
4375 \group_end:
4376 }
4377 }

```

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.

```

4378 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4379 {
4380   \bool_if:NTF \l_@@_initial_open_bool
4381   {
4382     \@@_open_x_initial_dim:
4383     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4384     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4385   }
4386   { \@@_set_initial_coords_from_anchor:n { base-east } }
4387   \bool_if:NTF \l_@@_final_open_bool
4388   {
4389     \@@_open_x_final_dim:
4390     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4391     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4392   }
4393   { \@@_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.

```

4394   \bool_lazy_all:nTF
4395   {
4396     \l_@@_initial_open_bool
4397     \l_@@_final_open_bool
4398     { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4399   }
4400   {
4401     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4402     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4403   }

```

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 (?).

```

4404   {
4405     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4406     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4407   }
4408   \@@_draw_line:
4409 }

```

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.

```

4410 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4411 {
4412   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4413   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4414   {
4415     \@@_find_extremities_of_line: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.

```

4416 \group_begin:
4417 \@@_open_shorten:
4418 \int_if_zero:nTF { #1 }
4419 { \color { nicematrix-first-row } }
4420 {

```

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.

```

4421 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4422 { \color { nicematrix-last-row } }
4423 }
4424 \keys_set:nn { nicematrix / xdots } { #3 }
4425 \@@_color:o \l_@@_xdots_color_tl
4426 \@@_actually_draw_Cdots:
4427 \group_end:
4428 }
4429 }

```

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`.

```

4430 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4431 {
4432 \bool_if:NTF \l_@@_initial_open_bool
4433 { \@@_open_x_initial_dim: }
4434 { \@@_set_initial_coords_from_anchor:n { mid-east } }
4435 \bool_if:NTF \l_@@_final_open_bool
4436 { \@@_open_x_final_dim: }
4437 { \@@_set_final_coords_from_anchor:n { mid-west } }
4438 \bool_lazy_and:nnTF
4439 { \l_@@_initial_open_bool }
4440 { \l_@@_final_open_bool }
4441 {
4442 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4443 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4444 \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4445 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4446 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4447 }
4448 {
4449 \bool_if:NT \l_@@_initial_open_bool
4450 { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4451 \bool_if:NT \l_@@_final_open_bool
4452 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4453 }
4454 \@@_draw_line:
4455 }
4456 \cs_new_protected:Npn \@@_open_y_initial_dim:
4457 {
4458 \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4459 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4460 {

```

```

4461 \cs_if_exist:cT
4462 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4463 {
4464     \pgfpointanchor
4465     { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4466     { north }
4467     \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4468     { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4469 }
4470 }
4471 \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4472 {
4473     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4474     \dim_set:Nn \l_@@_y_initial_dim
4475     {
4476         \fp_to_dim:n
4477         {
4478             \pgf@y
4479             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4480         }
4481     }
4482 }
4483 }
4484 \cs_new_protected:Npn \@@_open_y_final_dim:
4485 {
4486     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4487     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4488     {
4489         \cs_if_exist:cT
4490         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4491         {
4492             \pgfpointanchor
4493             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4494             { south }
4495             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4496             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4497         }
4498     }
4499     \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4500     {
4501         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4502         \dim_set:Nn \l_@@_y_final_dim
4503         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4504     }
4505 }

```

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.

```

4506 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4507 {
4508     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4509     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4510     {
4511         \@@_find_extremities_of_line: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.

```

4512 \group_begin:
4513     \@@_open_shorten:
4514     \int_if_zero:nTF { #2 }
4515     { \color { nicematrix-first-col } }
4516     {
4517         \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4518         { \color { nicematrix-last-col } }

```

```

4519     }
4520     \keys_set:nn { nicematrix / xdots } { #3 }
4521     \@@_color:o \l_@@_xdots_color_tl
4522     \@@_actually_draw_Vdots:
4523     \group_end:
4524   }
4525 }

```

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.`

The following function is also used by `\Vdotsfor`.

```

4526 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4527 {

```

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

```

4528   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }

```

We have to determine the  $x$ -value of the vertical rule that we will have to draw.

```

4529   {
4530     \@@_open_y_initial_dim:
4531     \@@_open_y_final_dim:
4532     \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4533     {
4534       \@@_qpoint:n { col - 1 }
4535       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4536       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4537       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4538       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4539     }
4540     {
4541       \bool_lazy_and:nnTF
4542       { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4543       {
4544         \int_compare_p:nNn
4545         { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4546       }

```

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

```

4547     {
4548       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4549       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4550       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4551       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4552       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4553     }

```

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

```

4554     {
4555       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4556       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4557       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4558       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }

```



```

4559     }
4560   }
4561 }

```

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

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

```

4562 {
4563   \bool_set_false:N \l_tmpa_bool
4564   \bool_if:NF \l_@@_initial_open_bool
4565   {
4566     \bool_if:NF \l_@@_final_open_bool
4567     {
4568       \@@_set_initial_coords_from_anchor:n { south-west }
4569       \@@_set_final_coords_from_anchor:n { north-west }
4570       \bool_set:Nn \l_tmpa_bool
4571       {
4572         \dim_compare_p:nNn
4573         { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4574       }
4575     }
4576   }

```

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

```

4577   \bool_if:NTF \l_@@_initial_open_bool
4578   {
4579     \@@_open_y_initial_dim:
4580     \@@_set_final_coords_from_anchor:n { north }
4581     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4582   }
4583   {
4584     \@@_set_initial_coords_from_anchor:n { south }
4585     \bool_if:NTF \l_@@_final_open_bool
4586     { \@@_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.

```

4587   {
4588     \@@_set_final_coords_from_anchor:n { north }
4589     \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4590     {
4591       \dim_set:Nn \l_@@_x_initial_dim
4592       {
4593         \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4594         \l_@@_x_initial_dim \l_@@_x_final_dim
4595       }
4596     }
4597   }
4598 }
4599 }
4600 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4601 \@@_draw_line:
4602 }

```

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.

```

4603 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4604 {
4605   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4606   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4607   {
4608     \@@_find_extremities_of_line: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.

```

4609     \group_begin:
4610     \@@_open_shorten:
4611     \keys_set:nn { nicematrix / xdots } { #3 }
4612     \@@_color:o \l_@@_xdots_color_tl
4613     \@@_actually_draw_Ddots:
4614     \group_end:
4615   }
4616 }

```

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.`

```

4617 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4618 {
4619   \bool_if:NTF \l_@@_initial_open_bool
4620   {
4621     \@@_open_y_initial_dim:
4622     \@@_open_x_initial_dim:
4623   }
4624   { \@@_set_initial_coords_from_anchor:n { south-east } }
4625   \bool_if:NTF \l_@@_final_open_bool
4626   {
4627     \@@_open_x_final_dim:
4628     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4629   }
4630   { \@@_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.

```

4631   \bool_if:NT \l_@@_parallelize_diags_bool
4632   {
4633     \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).

```

4634     \int_compare:nNnTF { \g_@@_ddots_int } = { \c_one_int }

```

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.

```

4635     {
4636       \dim_gset:Nn \g_@@_delta_x_one_dim
4637       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4638       \dim_gset:Nn \g_@@_delta_y_one_dim
4639       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4640     }

```

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`.

```

4641     {
4642       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4643       {
4644         \dim_set:Nn \l_@@_y_final_dim

```

```

4645         {
4646             \l_@@_y_initial_dim +
4647             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4648             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4649         }
4650     }
4651 }
4652 }
4653 \@@_draw_line:
4654 }

```

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.

```

4655 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4656 {
4657     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4658     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4659     {
4660         \@@_find_extremities_of_line: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.

```

4661     \group_begin:
4662     \@@_open_shorten:
4663     \keys_set:nn { nicematrix / xdots } { #3 }
4664     \@@_color:o \l_@@_xdots_color_tl
4665     \@@_actually_draw_Iddots:
4666     \group_end:
4667 }
4668 }

```

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.

```

4669 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4670 {
4671     \bool_if:NTF \l_@@_initial_open_bool
4672     {
4673         \@@_open_y_initial_dim:
4674         \@@_open_x_initial_dim:
4675     }
4676     { \@@_set_initial_coords_from_anchor:n { south-west } }
4677     \bool_if:NTF \l_@@_final_open_bool
4678     {
4679         \@@_open_y_final_dim:
4680         \@@_open_x_final_dim:
4681     }
4682     { \@@_set_final_coords_from_anchor:n { north-east } }
4683     \bool_if:NT \l_@@_parallelize_diags_bool
4684     {
4685         \int_gincr:N \g_@@_iddots_int
4686         \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }

```

```

4687     {
4688         \dim_gset:Nn \g_@@_delta_x_two_dim
4689         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4690         \dim_gset:Nn \g_@@_delta_y_two_dim
4691         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4692     }
4693     {
4694         \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4695         {
4696             \dim_set:Nn \l_@@_y_final_dim
4697             {
4698                 \l_@@_y_initial_dim +
4699                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4700                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4701             }
4702         }
4703     }
4704 }
4705 \@@_draw_line:
4706 }

```

## 17 The actual instructions for drawing the dotted lines with Tikz

The command `\@@_draw_line:` should be used in a `{pgfpicture}`. It 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`

```

4707 \cs_new_protected:Npn \@@_draw_line:
4708 {
4709     \pgfrememberpicturepositiononpagetrue
4710     \pgf@relevantforpicturesizefalse
4711     \bool_lazy_or:nnTF
4712     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4713     { \l_@@_dotted_bool }
4714     { \@@_draw_standard_dotted_line: }
4715     { \@@_draw_unstandard_dotted_line: }
4716 }

```

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.

```

4717 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4718 {
4719     \begin { scope }
4720     \@@_draw_unstandard_dotted_line:o
4721     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4722 }

```

We have used the fact that, in PGF, un 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.

```

4723 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4724 {
4725   \@@_draw_unstandard_dotted_line:nooo
4726   { #1 }
4727   \l_@@_xdots_up_tl
4728   \l_@@_xdots_down_tl
4729   \l_@@_xdots_middle_tl
4730 }
4731 \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.

```

4732 \hook_gput_code:nnn { begindocument } { . }
4733 {
4734   \IfPackageLoadedT { tikz }
4735   {
4736     \tikzset
4737     {
4738       @@_node_above / .style = { sloped , above } ,
4739       @@_node_below / .style = { sloped , below } ,
4740       @@_node_middle / .style =
4741       {
4742         sloped ,
4743         inner~sep = \c_@@_innersep_middle_dim
4744       }
4745     }
4746   }
4747 }

4748 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4749 {

```

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.

```

4750 \dim_zero_new:N \l_@@_l_dim
4751 \dim_set:Nn \l_@@_l_dim
4752 {
4753   \fp_to_dim:n
4754   {
4755     sqrt
4756     (
4757       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4758       +
4759       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4760     )
4761   }
4762 }

```

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.

```

4763 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4764 {
4765   \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }

```

```

4766 \@@_draw_unstandard_dotted_line_i:
4767 }

```

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

```

4768 \bool_if:NT \l_@@_xdots_h_labels_bool
4769 {
4770   \tikzset
4771   {
4772     @@_node_above / .style = { auto = left } ,
4773     @@_node_below / .style = { auto = right } ,
4774     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4775   }
4776 }
4777 \tl_if_empty:nF { #4 }
4778 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4779 \draw
4780 [ #1 ]
4781 ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4782 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4783 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4784 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4785 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4786 \end { scope }
4787 }
4788 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4789 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4790 {
4791   \dim_set:Nn \l_tmpa_dim
4792   {
4793     \l_@@_x_initial_dim
4794     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4795     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4796   }
4797   \dim_set:Nn \l_tmpb_dim
4798   {
4799     \l_@@_y_initial_dim
4800     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4801     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4802   }
4803   \dim_set:Nn \l_@@_tmpc_dim
4804   {
4805     \l_@@_x_final_dim
4806     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4807     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4808   }
4809   \dim_set:Nn \l_@@_tmpd_dim
4810   {
4811     \l_@@_y_final_dim
4812     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4813     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4814   }
4815   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4816   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4817   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4818   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4819 }

```

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

```

4820 \cs_new_protected:Npn \@@_draw_standard_dotted_line:

```

```

4821 {
4822   \group_begin:
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.
4823   \dim_zero_new:N \l_@@_l_dim
4824   \dim_set:Nn \l_@@_l_dim
4825   {
4826     \fp_to_dim:n
4827     {
4828       sqrt
4829       (
4830         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4831         +
4832         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4833       )
4834     }
4835   }

```

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.

```

4836   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4837   {
4838     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4839     { \@@_draw_standard_dotted_line_i: }
4840   }
4841   \group_end:
4842   \bool_lazy_all:nF
4843   {
4844     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4845     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4846     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4847   }
4848   { \@@_labels_standard_dotted_line: }
4849 }
4850 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4851 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4852 {

```

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

```

4853   \int_set:Nn \l_tmpa_int
4854   {
4855     \dim_ratio:nn
4856     {
4857       \l_@@_l_dim
4858       - \l_@@_xdots_shorten_start_dim
4859       - \l_@@_xdots_shorten_end_dim
4860     }
4861     { \l_@@_xdots_inter_dim }
4862   }

```

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

```

4863   \dim_set:Nn \l_tmpa_dim
4864   {
4865     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4866     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4867   }
4868   \dim_set:Nn \l_tmpb_dim
4869   {
4870     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *

```

```

4871     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4872 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4873 \dim_gadd:Nn \l_@@_x_initial_dim
4874 {
4875   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4876   \dim_ratio:nn
4877   {
4878     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4879     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4880   }
4881   { 2 \l_@@_l_dim }
4882 }
4883 \dim_gadd:Nn \l_@@_y_initial_dim
4884 {
4885   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4886   \dim_ratio:nn
4887   {
4888     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4889     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4890   }
4891   { 2 \l_@@_l_dim }
4892 }
4893 \pgf@relevantforpicturesizefalse
4894 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4895 {
4896   \pgfpathcircle
4897   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4898   { \l_@@_xdots_radius_dim }
4899   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4900   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4901 }
4902 \pgfusepathqfill
4903 }

```

```

4904 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4905 {
4906   \pgfscope
4907   \pgftransformshift
4908   {
4909     \pgfpointlineattime { 0.5 }
4910     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4911     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4912   }
4913   \fp_set:Nn \l_tmpa_fp
4914   {
4915     atand
4916     (
4917       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4918       \l_@@_x_final_dim - \l_@@_x_initial_dim
4919     )
4920   }
4921   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4922   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4923   \tl_if_empty:NF \l_@@_xdots_middle_tl
4924   {
4925     \begin { pgfscope }
4926     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4927     \pgfnode
4928     { rectangle }
4929     { center }

```



```

4930     {
4931         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4932         {
4933             \c_math_toggle_token
4934             \scriptstyle \l_@@_xdots_middle_tl
4935             \c_math_toggle_token
4936         }
4937     }
4938     { }
4939     {
4940         \pgfsetfillcolor { white }
4941         \pgfusepath { fill }
4942     }
4943     \end { pgfscope }
4944 }
4945 \tl_if_empty:NF \l_@@_xdots_up_tl
4946 {
4947     \pgfnode
4948     { rectangle }
4949     { south }
4950     {
4951         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4952         {
4953             \c_math_toggle_token
4954             \scriptstyle \l_@@_xdots_up_tl
4955             \c_math_toggle_token
4956         }
4957     }
4958     { }
4959     { \pgfusepath { } }
4960 }
4961 \tl_if_empty:NF \l_@@_xdots_down_tl
4962 {
4963     \pgfnode
4964     { rectangle }
4965     { north }
4966     {
4967         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4968         {
4969             \c_math_toggle_token
4970             \scriptstyle \l_@@_xdots_down_tl
4971             \c_math_toggle_token
4972         }
4973     }
4974     { }
4975     { \pgfusepath { } }
4976 }
4977 \endpgfscope
4978 }

```

## 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 `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```
4979 \hook_gput_code:nnn { begindocument } { . }
4980 {
```

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

```
4981 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4982 \cs_new_protected:Npn \@@_Ldots:
4983 { \@@_collect_options:n { \@@_Ldots_i } }
4984 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4985 {
4986   \int_if_zero:nTF { \c@jCol }
4987   { \@@_error:nn { in~first~col } { \Ldots } }
4988   {
4989     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
4990     { \@@_error:nn { in~last~col } { \Ldots } }
4991     {
4992       \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
4993       { #1 , down = #2 , up = #3 , middle = #4 }
4994     }
4995   }
4996   \bool_if:NF \l_@@_nullify_dots_bool
4997   { \phantom { \ensuremath { \@@_old_ldots: } } }
4998   \bool_gset_true:N \g_@@_empty_cell_bool
4999 }
```

```
5000 \cs_new_protected:Npn \@@_Cdots:
5001 { \@@_collect_options:n { \@@_Cdots_i } }
5002 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5003 {
5004   \int_if_zero:nTF { \c@jCol }
5005   { \@@_error:nn { in~first~col } { \Cdots } }
5006   {
5007     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5008     { \@@_error:nn { in~last~col } { \Cdots } }
5009     {
5010       \@@_instruction_of_type:nnn { \c_false_bool } { \Cdots }
5011       { #1 , down = #2 , up = #3 , middle = #4 }
5012     }
5013   }
5014   \bool_if:NF \l_@@_nullify_dots_bool
5015   { \phantom { \ensuremath { \@@_old_cdots: } } }
5016   \bool_gset_true:N \g_@@_empty_cell_bool
5017 }
```

```
5018 \cs_new_protected:Npn \@@_Vdots:
5019 { \@@_collect_options:n { \@@_Vdots_i } }
5020 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5021 {
5022   \int_if_zero:nTF { \c@iRow }
5023   { \@@_error:nn { in~first~row } { \Vdots } }
5024   {
5025     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5026     { \@@_error:nn { in~last~row } { \Vdots } }
5027     {
5028       \@@_instruction_of_type:nnn { \c_false_bool } { \Vdots }
5029       { #1 , down = #2 , up = #3 , middle = #4 }
5030     }
5031   }
5032   \bool_if:NF \l_@@_nullify_dots_bool
5033   { \phantom { \ensuremath { \@@_old_vdots: } } }
```

```

5034     \bool_gset_true:N \g_@@_empty_cell_bool
5035 }

5036 \cs_new_protected:Npn \@@_Ddots:
5037 { \@@_collect_options:n { \@@_Ddots_i } }
5038 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5039 {
5040     \int_case:nnF \c@iRow
5041     {
5042         0 { \@@_error:nn { in~first~row } { \Ddots } }
5043         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5044     }
5045     {
5046         \int_case:nnF \c@jCol
5047         {
5048             0 { \@@_error:nn { in~first~col } { \Ddots } }
5049             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5050         }
5051         {
5052             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5053             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5054             { #1 , down = #2 , up = #3 , middle = #4 }
5055         }
5056     }
5057 }
5058 \bool_if:NF \l_@@_nullify_dots_bool
5059 { \phantom { \ensuremath { \@@_old_ddots: } } }
5060 \bool_gset_true:N \g_@@_empty_cell_bool
5061 }

5062 \cs_new_protected:Npn \@@_Iddots:
5063 { \@@_collect_options:n { \@@_Iddots_i } }
5064 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5065 {
5066     \int_case:nnF \c@iRow
5067     {
5068         0 { \@@_error:nn { in~first~row } { \Iddots } }
5069         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5070     }
5071     {
5072         \int_case:nnF \c@jCol
5073         {
5074             0 { \@@_error:nn { in~first~col } { \Iddots } }
5075             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5076         }
5077         {
5078             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5079             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5080             { #1 , down = #2 , up = #3 , middle = #4 }
5081         }
5082     }
5083     \bool_if:NF \l_@@_nullify_dots_bool
5084     { \phantom { \ensuremath { \@@_old_iddots: } } }
5085     \bool_gset_true:N \g_@@_empty_cell_bool
5086 }
5087 }

```

End of the \AddToHook.

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

```

5088 \keys_define:nn { nicematrix / Ddots }
5089 {

```

```

5090     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5091     draw-first .default:n = true ,
5092     draw-first .value_forbidden:n = true
5093 }

```

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

```

5094 \cs_new_protected:Npn \@@_Hspace:
5095 {
5096   \bool_gset_true:N \g_@@_empty_cell_bool
5097   \hspace
5098 }

```

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`.

```

5099 \cs_set_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`.

```

5100 \cs_new:Npn \@@_Hdotsfor:
5101 {
5102   \bool_lazy_and:nnTF
5103     { \int_if_zero_p:n { \c@jCol } }
5104     { \int_if_zero_p:n { \l_@@_first_col_int } }
5105     {
5106       \bool_if:NTF \g_@@_after_col_zero_bool
5107       {
5108         \multicolumn { 1 } { c } { }
5109         \@@_Hdotsfor_i:
5110       }
5111       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5112     }
5113   {
5114     \multicolumn { 1 } { c } { }
5115     \@@_Hdotsfor_i:
5116   }
5117 }

```

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:`).

```

5118 \hook_gput_code:nnn { begindocument } { . }
5119 {

```

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

```

5120   \cs_new_protected:Npn \@@_Hdotsfor_i:
5121     { \@@_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`).

```

5122   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5123   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5124     {
5125     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5126     {
5127       \@@_Hdotsfor:nnnn
5128       { \int_use:N \c@iRow }
5129       { \int_use:N \c@jCol }
5130       { #2 }
5131       {
5132         #1 , #3 ,

```

```

5133         down = \exp_not:n { #4 } ,
5134         up = \exp_not:n { #5 } ,
5135         middle = \exp_not:n { #6 }
5136     }
5137 }
5138 \prg_replicate:nn { #2 - 1 }
5139 {
5140     &
5141     \multicolumn { 1 } { c } { }
5142     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5143 }
5144 }
5145 }

```

```

5146 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5147 {
5148     \bool_set_false:N \l_@@_initial_open_bool
5149     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5150     \int_set:Nn \l_@@_initial_i_int { #1 }
5151     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5152     \int_compare:nNnTF { #2 } = { \c_one_int }
5153     {
5154         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5155         \bool_set_true:N \l_@@_initial_open_bool
5156     }
5157     {
5158         \cs_if_exist:cTF
5159         {
5160             pgf @ sh @ ns @ \@@_env:
5161             - \int_use:N \l_@@_initial_i_int
5162             - \int_eval:n { #2 - 1 }
5163         }
5164         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5165         {
5166             \int_set:Nn \l_@@_initial_j_int { #2 }
5167             \bool_set_true:N \l_@@_initial_open_bool
5168         }
5169     }
5170     \int_compare:nNnTF { #2 + #3 - 1 } = { \c_jCol }
5171     {
5172         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5173         \bool_set_true:N \l_@@_final_open_bool
5174     }
5175     {
5176         \cs_if_exist:cTF
5177         {
5178             pgf @ sh @ ns @ \@@_env:
5179             - \int_use:N \l_@@_final_i_int
5180             - \int_eval:n { #2 + #3 }
5181         }
5182         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5183         {
5184             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5185             \bool_set_true:N \l_@@_final_open_bool
5186         }
5187     }
5188     \group_begin:
5189     \@@_open_shorten:
5190     \int_if_zero:nTF { #1 }
5191     { \color { nicematrix-first-row } }

```

```

5192     {
5193         \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5194         { \color { nicematrix-last-row } }
5195     }
5196     \keys_set:nn { nicematrix / xdots } { #4 }
5197     \@@_color:o \l_@@_xdots_color_tl
5198     \@@_actually_draw_Ldots:
5199     \group_end:

```

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_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5200     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5201     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5202 }

```

```

5203 \hook_gput_code:nnn { begindocument } { . }
5204 {
5205     \cs_new_protected:Npn \@@_Vdotsfor:
5206     { \@@_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`).

```

5207     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } { } } }
5208     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5209     {
5210         \bool_gset_true:N \g_@@_empty_cell_bool
5211         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5212         {
5213             \@@_Vdotsfor:nnnn
5214             { \int_use:N \c@iRow }
5215             { \int_use:N \c@jCol }
5216             { #2 }
5217             {
5218                 #1 , #3 ,
5219                 down = \exp_not:n { #4 } ,
5220                 up = \exp_not:n { #5 } ,
5221                 middle = \exp_not:n { #6 }
5222             }
5223         }
5224     }
5225 }

```

```

5226 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5227 {
5228     \bool_set_false:N \l_@@_initial_open_bool
5229     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5230     \int_set:Nn \l_@@_initial_j_int { #2 }
5231     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5232     \int_compare:nNnTF { #1 } = { \c_one_int }
5233     {
5234         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5235         \bool_set_true:N \l_@@_initial_open_bool
5236     }
5237     {
5238         \cs_if_exist:cTF
5239         {
5240             pgf @ sh @ ns @ \@@_env:

```

```

5241         - \int_eval:n { #1 - 1 }
5242         - \int_use:N \l_@@_initial_j_int
5243     }
5244     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5245     {
5246         \int_set:Nn \l_@@_initial_i_int { #1 }
5247         \bool_set_true:N \l_@@_initial_open_bool
5248     }
5249 }
5250 \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5251 {
5252     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5253     \bool_set_true:N \l_@@_final_open_bool
5254 }
5255 {
5256     \cs_if_exist:cTF
5257     {
5258         pgf @ sh @ ns @ \@@_env:
5259         - \int_eval:n { #1 + #3 }
5260         - \int_use:N \l_@@_final_j_int
5261     }
5262     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5263     {
5264         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5265         \bool_set_true:N \l_@@_final_open_bool
5266     }
5267 }
5268 \group_begin:
5269 \@@_open_shorten:
5270 \int_if_zero:nTF { #2 }
5271 { \color { nicematrix-first-col } }
5272 {
5273     \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5274     { \color { nicematrix-last-col } }
5275 }
5276 \keys_set:nn { nicematrix / xdots } { #4 }
5277 \@@_color:o \l_@@_xdots_color_tl
5278 \@@_actually_draw_Vdots:
5279 \group_end:

```

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_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5280     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5281     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5282 }

```

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

```

5283 \NewDocumentCommand \@@_rotate: { 0 { } }
5284 {
5285     \bool_gset_true:N \g_@@_rotate_bool
5286     \keys_set:nn { nicematrix / rotate } { #1 }
5287     \ignorespaces
5288 }
5289 \keys_define:nn { nicematrix / rotate }
5290 {
5291     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5292     c .value_forbidden:n = true ,
5293     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5294 }

```

## 19 The command `\line` accessible in `code-after`

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>

```

5295 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5296 {
5297   \tl_if_empty:nTF { #2 }
5298   { #1 }
5299   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5300 }
5301 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5302 { \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).

```

5303 \hook_gput_code:nnn { begindocument } { . }
5304 {
```

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

```

5305   \tl_set_rescan:Nnn \l_tmpa_tl { }
5306   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5307   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5308   {
5309     \group_begin:
5310     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5311     \@@_color:o \l_@@_xdots_color_tl
5312     \use:e
5313     {
5314       \@@_line_i:nn
5315       { \@@_double_int_eval:n #2 - \q_stop }
5316       { \@@_double_int_eval:n #3 - \q_stop }
5317     }
5318     \group_end:
5319   }
5320 }
5321 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5322 {
5323   \bool_set_false:N \l_@@_initial_open_bool
5324   \bool_set_false:N \l_@@_final_open_bool
5325   \bool_lazy_or:nnTF
5326   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5327   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5328   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```

5329   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5330 }
```

<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`.



```

5331 \hook_gput_code:nnn { begindocument } { . }
5332 {
5333   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5334   {

```

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:`.

```

5335     \c_@@_pgfortikzpicture_tl
5336     \@@_draw_line_iii:nn { #1 } { #2 }
5337     \c_@@_endpgfortikzpicture_tl
5338   }
5339 }

```

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

```

5340 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5341 {
5342   \pgfrememberpicturepositiononpagetrue
5343   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5344   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5345   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5346   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5347   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5348   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5349   \@@_draw_line:
5350 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

**#1** is the first row *after* the scope of the instructions in **#2**

However, both arguments are implicit because they are taken by curryfication.

```

5351 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNt { \c@iRow } < }
5352 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNf { \c@jCol } < }

```

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

```

5353 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5354 {
5355   \tl_gput_right:Ne \g_@@_row_style_tl
5356   {

```

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

```

5357     \exp_not:N
5358     \@@_if_row_less_than:nn
5359     { \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`).

```

5360     {
5361         \exp_not:N
5362         \@@_if_col_greater_than:nn
5363         { \int_eval:n { \c@jCol } }
5364         { \exp_not:n { #1 } \scan_stop: }
5365     }
5366 }
5367 }
5368 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5369 \keys_define:nn { nicematrix / RowStyle }
5370 {
5371     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5372     cell-space-top-limit .value_required:n = true ,
5373     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5374     cell-space-bottom-limit .value_required:n = true ,
5375     cell-space-limits .meta:n =
5376     {
5377         cell-space-top-limit = #1 ,
5378         cell-space-bottom-limit = #1 ,
5379     } ,
5380     color .tl_set:N = \l_@@_color_tl ,
5381     color .value_required:n = true ,
5382     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5383     bold .default:n = true ,
5384     nb-rows .code:n =
5385     \str_if_eq:eeTF { #1 } { * }
5386     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5387     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5388     nb-rows .value_required:n = true ,
5389     fill .tl_set:N = \l_@@_fill_tl ,
5390     fill .value_required:n = true ,
5391     opacity .tl_set:N = \l_@@_opacity_tl ,
5392     opacity .value_required:n = true ,
5393     rowcolor .tl_set:N = \l_@@_fill_tl ,
5394     rowcolor .value_required:n = true ,
5395     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5396     rounded-corners .default:n = 4 pt ,
5397     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5398 }

5399 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5400 {
5401     \group_begin:
5402     \tl_clear:N \l_@@_fill_tl
5403     \tl_clear:N \l_@@_opacity_tl
5404     \tl_clear:N \l_@@_color_tl
5405     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5406     \dim_zero:N \l_@@_rounded_corners_dim
5407     \dim_zero:N \l_tmpa_dim
5408     \dim_zero:N \l_tmpb_dim
5409     \keys_set:nn { nicematrix / RowStyle } { #1 }

5410     \tl_if_empty:NF \l_@@_fill_tl
5411     {
5412         \@@_add_opacity_to_fill:
5413         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5414         {

```

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

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

```

5415         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5416         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5417         {
5418             \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5419             - *
5420         }
5421         { \dim_use:N \l_@@_rounded_corners_dim }
5422     }
5423 }
5424 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5425     \dim_compare:nNt { \l_tmpa_dim } > { \c_zero_dim }
5426     {
5427         \@@_put_in_row_style:e
5428         {
5429             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5430             {

```

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

```

5431         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5432         { \dim_use:N \l_tmpa_dim }
5433     }
5434 }
5435 }

```

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

```

5436     \dim_compare:nNt { \l_tmpb_dim } > { \c_zero_dim }
5437     {
5438         \@@_put_in_row_style:e
5439         {
5440             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5441             {
5442                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5443                 { \dim_use:N \l_tmpb_dim }
5444             }
5445         }
5446     }

```

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

```

5447     \tl_if_empty:NF \l_@@_color_tl
5448     {
5449         \@@_put_in_row_style:e
5450         {
5451             \mode_leave_vertical:
5452             \@@_color:n { \l_@@_color_tl }
5453         }
5454     }

```

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

```

5455     \bool_if:NT \l_@@_bold_row_style_bool
5456     {
5457         \@@_put_in_row_style:n
5458         {
5459             \exp_not:n
5460             {
5461                 \if_mode_math:
5462                     \c_math_toggle_token
5463                     \bfseries \boldmath
5464                     \c_math_toggle_token
5465                 \else:
5466                     \bfseries \boldmath
5467                 \fi:
5468             }

```

```

5469     }
5470   }
5471   \group_end:
5472   \g_@@_row_style_tl
5473   \ignorespaces
5474 }

```

The following commande must *not* be protected.

```

5475 \cs_new:Npn \@@_rounded_from_row:n #1
5476 {
5477   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5478   { \int_eval:n { #1 } - 1 }
5479   {
5480     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5481     - \exp_not:n { \int_use:N \c@jCol }
5482   }
5483   { \dim_use:N \l_@@_rounded_corners_dim }
5484 }

```

## 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).

```

5485 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5486 {

```

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.

```

5487   \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`.

```

5488   \str_if_in:nnF { #1 } { !! }
5489   {
5490     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

5491     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5492   }
5493   \int_if_zero:nTF { \l_tmpa_int }

```

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

```

5494   {
5495     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5496     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5497   }

```

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

```

5498     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5499   }
5500   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5501   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

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

```

5502   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5503   {
5504     \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5505     {

```

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

```

5506     \group_begin:
5507     \pgfsetcornersarced
5508     {
5509       \pgfpoint
5510       { \l_@@_tab_rounded_corners_dim }
5511       { \l_@@_tab_rounded_corners_dim }
5512     }

```

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.

```

5513     \bool_if:NTF \l_@@_hvlines_bool
5514     {
5515       \pgfpathrectanglecorners
5516       {
5517         \pgfpointadd
5518         { \@@_qpoint:n { row-1 } }
5519         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5520       }
5521       {
5522         \pgfpointadd
5523         {
5524           \@@_qpoint:n
5525           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5526         }
5527         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5528       }
5529     }
5530     {
5531       \pgfpathrectanglecorners
5532       { \@@_qpoint:n { row-1 } }
5533       {
5534         \pgfpointadd
5535         {
5536           \@@_qpoint:n
5537           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5538         }

```

```

5539         { \pgfpoint \c_zero_dim \arrayrulewidth }
5540     }
5541 }
5542 \pgfusepath { clip }
5543 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5544 }
5545 }

```

The macro \@@\_actually\_color: will actually fill all the rectangles, color by color (using the sequence \l\_@@\_colors\_seq and all the token lists of the form \l\_@@\_color\_i\_tl).

```

5546 \cs_new_protected:Npn \@@_actually_color:
5547 {
5548     \pgfpicture
5549     \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.

```

5550     \@@_clip_with_rounded_corners:
5551     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5552     {
5553         \int_compare:nNnTF { ##1 } = { \c_one_int }
5554         {
5555             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5556             \use:c { g_@@_color _ 1 _tl }
5557             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5558         }
5559         {
5560             \begin { pgfscope }
5561                 \@@_color_opacity: ##2
5562                 \use:c { g_@@_color _ ##1 _tl }
5563                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5564                 \pgfusepath { fill }
5565             \end { pgfscope }
5566         }
5567     }
5568 \endpgfpicture
5569 }

```

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

```

5570 \cs_new_protected:Npn \@@_color_opacity:
5571 {
5572     \peek_meaning:NTF [
5573         { \@@_color_opacity:w }
5574         { \@@_color_opacity:w [ ] }
5575     }

```

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.

```

5576 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5577 {
5578     \tl_clear:N \l_tmpa_tl
5579     \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.

```

5580     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5581     \tl_if_empty:NTF \l_tmpb_tl
5582     { \@declaredcolor }
5583     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5584 }

```

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

```

5585 \keys_define:nn { nicematrix / color-opacity }
5586 {
5587   opacity .tl_set:N          = \l_tmpa_tl ,
5588   opacity .value_required:n = true
5589 }

```

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

```

5590 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5591 {
5592   \def \l_@@_rows_tl { #1 }
5593   \def \l_@@_cols_tl { #2 }
5594   \@@_cartesian_path:
5595 }

```

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

```

5596 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5597 {
5598   \tl_if_blank:nF { #2 }
5599   {
5600     \@@_add_to_colors_seq:en
5601     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5602     { \@@_cartesian_color:nn { #3 } { - } }
5603   }
5604 }

```

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

```

5605 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5606 {
5607   \tl_if_blank:nF { #2 }
5608   {
5609     \@@_add_to_colors_seq:en
5610     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5611     { \@@_cartesian_color:nn { - } { #3 } }
5612   }
5613 }

```

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

```

5614 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5615 {
5616   \tl_if_blank:nF { #2 }
5617   {
5618     \@@_add_to_colors_seq:en
5619     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5620     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5621   }
5622 }

```

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

```

5623 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5624 {
5625   \tl_if_blank:nF { #2 }
5626   {
5627     \@@_add_to_colors_seq:en
5628     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5629     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5630   }
5631 }

```

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

```

5632 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5633 {
5634   \@@_cut_on_hyphen:w #1 \q_stop
5635   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5636   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5637   \@@_cut_on_hyphen:w #2 \q_stop
5638   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5639   \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`.

```

5640   \@@_cartesian_path:n { #3 }
5641 }

```

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

```

5642 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5643 {
5644   \clist_map_inline:nn { #3 }
5645     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5646 }

```

```

5647 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5648 {
5649   \int_step_inline:nn { \c@iRow }
5650   {
5651     \int_step_inline:nn { \c@jCol }
5652     {
5653       \int_if_even:nTF { ####1 + ##1 }
5654         { \@@_cellcolor [ #1 ] { #2 } }
5655         { \@@_cellcolor [ #1 ] { #3 } }
5656       { ##1 - ####1 }
5657     }
5658   }
5659 }

```

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”.

```

5660 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5661 {
5662   \@@_rectanglecolor [ #1 ] { #2 }
5663   { 1 - 1 }
5664   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5665 }

5666 \keys_define:nn { nicematrix / rowcolors }
5667 {
5668   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5669   respect-blocks .default:n = true ,
5670   cols .tl_set:N = \l_@@_cols_tl ,
5671   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5672   restart .default:n = true ,
5673   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5674 }

```

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 `\l_@@_rowcolors` appears as a special case of `\l_@@_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*.

```
5675 \NewDocumentCommand \l_@@_rowlistcolors { 0 { } m m 0 { } }
5676 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
5677 \group_begin:
5678 \seq_clear_new:N \l_@@_colors_seq
5679 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5680 \tl_clear_new:N \l_@@_cols_tl
5681 \tl_set:Nn \l_@@_cols_tl { - }
5682 \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).

```
5683 \int_zero_new:N \l_@@_color_int
5684 \int_set_eq:NN \l_@@_color_int \c_one_int
5685 \bool_if:NT \l_@@_respect_blocks_bool
5686 {
```

We don't want to take into account a block which is completely 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`).

```
5687 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5688 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5689 { \l_@@_not_in_exterior_p:nnnnn ##1 }
5690 }
5691 \pgfpicture
5692 \pgf@relevantforpicturesizefalse
```

**#2** is the list of intervals of rows.

```
5693 \clist_map_inline:nn { #2 }
5694 {
5695 \tl_set:Nn \l_tmpa_tl { ##1 }
5696 \tl_if_in:NnTF \l_tmpa_tl { - }
5697 { \l_@@_cut_on_hyphen:w ##1 \q_stop }
5698 { \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.

```
5699 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5700 \int_set:Nn \l_@@_color_int
5701 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5702 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5703 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5704 {
```

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

```
5705 \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).

```
5706 \bool_if:NT \l_@@_respect_blocks_bool
5707 {
5708 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5709 { \l_@@_intersect_our_row_p:nnnnn #####1 }
5710 \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`.

```
5711 }
5712 \tl_set:Nn \l_@@_rows_tl
5713 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```

5714         \tl_set:Nc \l_@@_color_tl
5715         {
5716             \@@_color_index:n
5717             {
5718                 \int_mod:nn
5719                 { \l_@@_color_int - 1 }
5720                 { \seq_count:N \l_@@_colors_seq }
5721                 + 1
5722             }
5723         }
5724     \tl_if_empty:NF \l_@@_color_tl
5725     {
5726         \@@_add_to_colors_seq:ee
5727         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5728         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5729     }
5730     \int_incr:N \l_@@_color_int
5731     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5732 }
5733 }
5734 \endpgfpicture
5735 \group_end:
5736 }

```

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.

```

5737 \cs_new:Npn \@@_color_index:n #1
5738 {

```

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

```

5739     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5740     { \@@_color_index:n { #1 - 1 } }
5741     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5742 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5743 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5744 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5745 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5746 {
5747     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5748     { \int_set:Nn \l_tmpb_int { #3 } }
5749 }

```

```

5750 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5751 {
5752     \int_if_zero:nTF { #4 }
5753     { \prg_return_false: }
5754     {
5755         \int_compare:nNnTF { #2 } > { \c@jCol }
5756         { \prg_return_false: }
5757         { \prg_return_true: }
5758     }
5759 }

```

The following command return true when the block intersects the row \l\_tmpa\_int.

```

5760 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5761 {
5762   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5763   { \prg_return_false: }
5764   {
5765     \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5766     { \prg_return_false: }
5767     { \prg_return_true: }
5768   }
5769 }

```

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).

```

5770 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5771 {
5772   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5773   {
5774     \bool_if:NTF \l_@@_nocolor_used_bool
5775     { \@@_cartesian_path_normal_ii: }
5776     {
5777       \clist_if_empty:NTF \l_@@_corners_cells_clist
5778       { \@@_cartesian_path_normal_i:n { #1 } }
5779       { \@@_cartesian_path_normal_ii: }
5780     }
5781   }
5782   { \@@_cartesian_path_normal_i:n { #1 } }
5783 }

```

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.

```

5784 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5785 {
5786   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5787 \clist_map_inline:Nn \l_@@_cols_tl
5788 {

```

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

```

5789   \def \l_tmpa_tl { ##1 }
5790   \tl_if_in:NnTF \l_tmpa_tl { - }
5791   { \@@_cut_on_hyphen:w ##1 \q_stop }
5792   { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5793   \tl_if_empty:NTF \l_tmpa_tl
5794   { \def \l_tmpa_tl { 1 } }
5795   {
5796     \str_if_eq:eeT \l_tmpa_tl { * }
5797     { \def \l_tmpa_tl { 1 } }
5798   }
5799   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5800   { \@@_error:n { Invalid~col~number } }
5801   \tl_if_empty:NTF \l_tmpb_tl
5802   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5803   {
5804     \str_if_eq:eeT \l_tmpb_tl { * }
5805     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5806   }

```

```

5807 \int_compare:nNt { \l_tmpb_tl } > { \g_@@_col_total_int }
5808 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5809 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5810 \@@_qpoint:n { col - \l_tmpa_tl }
5811 \int_compare:nNtF { \l_@@_first_col_int } = { \l_tmpa_tl }
5812 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5813 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5814 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5815 \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.

```

5816 \clist_map_inline:Nn \l_@@_rows_tl
5817 {
5818   \def \l_tmpa_tl { #####1 }
5819   \tl_if_in:NnTF \l_tmpa_tl { - }
5820   { \@@_cut_on_hyphen:w #####1 \q_stop }
5821   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5822   \tl_if_empty:NtF \l_tmpa_tl
5823   { \def \l_tmpa_tl { 1 } }
5824   {
5825     \str_if_eq:eeT \l_tmpa_tl { * }
5826     { \def \l_tmpa_tl { 1 } }
5827   }
5828   \tl_if_empty:NtF \l_tmpb_tl
5829   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5830   {
5831     \str_if_eq:eeT \l_tmpb_tl { * }
5832     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5833   }
5834   \int_compare:nNt { \l_tmpa_tl } > { \g_@@_row_total_int }
5835   { \@@_error:n { Invalid-row-number } }
5836   \int_compare:nNt { \l_tmpb_tl } > { \g_@@_row_total_int }
5837   { \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`.

```

5838 \cs_if_exist:cF
5839 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5840 {
5841   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5842   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5843   \@@_qpoint:n { row - \l_tmpa_tl }
5844   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5845   \pgfpathrectanglecorners
5846   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5847   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5848 }
5849 }
5850 }
5851 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5852 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5853 {
5854   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5855   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5856 \clist_map_inline:Nn \l_@@_cols_tl
5857 {
5858   \@@_qpoint:n { col - ##1 }
5859   \int_compare:nNtF { \l_@@_first_col_int } = { ##1 }
5860   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }

```

```

5861      { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5862      \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5863      \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5864      \clist_map_inline:Nn \l_@@_rows_tl
5865      {
5866        \@@_if_in_corner:nF { #####1 - ##1 }
5867        {
5868          \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5869          \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5870          \@@_qpoint:n { row - #####1 }
5871          \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5872          \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5873          {
5874            \pgfpathrectanglecorners
5875            { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5876            { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5877          }
5878        }
5879      }
5880    }
5881  }

```

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`).

```

5882 \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

```

5883 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5884 {
5885   \bool_set_true:N \l_@@_nocolor_used_bool
5886   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5887   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5888   \clist_map_inline:Nn \l_@@_rows_tl
5889   {
5890     \clist_map_inline:Nn \l_@@_cols_tl
5891     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5892   }
5893 }

```

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.

```

5894 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5895 {
5896   \clist_set_eq:NN \l_tmpa_clist #1
5897   \clist_clear:N #1
5898   \clist_map_inline:Nn \l_tmpa_clist
5899   {

```

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

```

5900   \def \l_tmpa_tl { ##1 }
5901   \tl_if_in:NnTF \l_tmpa_tl { - }
5902   { \@@_cut_on_hyphen:w ##1 \q_stop }
5903   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5904   \bool_lazy_or:nnT
5905   { \str_if_eq_p:ee \l_tmpa_tl { * } }

```

```

5906     { \tl_if_blank_p:o \l_tmpa_tl }
5907     { \def \l_tmpa_tl { 1 } }
5908     \bool_lazy_or:nnT
5909     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5910     { \tl_if_blank_p:o \l_tmpb_tl }
5911     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5912     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5913     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5914     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5915     { \clist_put_right:Nn #1 { ####1 } }
5916   }
5917 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5918 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5919 {
5920   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5921   {

```

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`).

```

5922     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5923     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5924   }
5925   \ignorespaces
5926 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5927 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5928 {
5929   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5930   {
5931     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5932     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5933     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5934   }
5935   \ignorespaces
5936 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5937 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5938 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5939 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5940 {

```

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`.

```

5941   \seq_gclear:N \g_tmpa_seq
5942   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5943   { \@@_rowlistcolors_tabular:nnnn ##1 }
5944   \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`.

```

5945 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5946 {
5947   { \int_use:N \c@iRow }
5948   { \exp_not:n { #1 } }
5949   { \exp_not:n { #2 } }
5950   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5951 }
5952 \ignorespaces
5953 }

```

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`).

```

5954 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5955 {
5956   \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).

```

5957   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5958   {
5959     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5960     {
5961       \@@_rowlistcolors
5962       [ \exp_not:n { #2 } ]
5963       { #1 - \int_eval:n { \c@iRow - 1 } }
5964       { \exp_not:n { #3 } }
5965       [ \exp_not:n { #4 } ]
5966     }
5967   }
5968 }

```

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.

```

5969 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5970 {
5971   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5972   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5973   \seq_gclear:N \g_@@_rowlistcolors_seq
5974 }

5975 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5976 {
5977   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5978   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5979 }

```

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.

```

5980 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5981 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```
5982 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
5983 {
```

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).

```
5984 \tl_gput_left:N \g_@@_pre_code_before_tl
5985 {
5986   \exp_not:N \columncolor [ #1 ]
5987   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5988 }
5989 }
5990 }

5991 \cs_new_protected:Npn \@@_EmptyColumn:n #1
5992 {
5993   \clist_map_inline:nn { #1 }
5994   {
5995     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5996     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
5997     \columncolor { nocolor } { ##1 }
5998   }
5999 }

6000 \cs_new_protected:Npn \@@_EmptyRow:n #1
6001 {
6002   \clist_map_inline:nn { #1 }
6003   {
6004     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6005     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6006     \rowcolor { nocolor } { ##1 }
6007   }
6008 }
```

## 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`.

```
6009 \cs_set_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`.

```
6010 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6011 {
6012   \int_if_zero:nTF { \l_@@_first_col_int }
6013   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6014   {
```



```

6015 \int_if_zero:nTF { \c@jCol }
6016 {
6017     \int_compare:nNnF { \c@iRow } = { -1 }
6018     {
6019         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6020         { #1 }
6021     }
6022 }
6023 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6024 }
6025 }

```

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.

```

6026 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6027 {
6028     \int_if_zero:nF { \c@iRow }
6029     {
6030         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6031         {
6032             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6033             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6034         }
6035     }
6036 }

```

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`.

```

6037 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6038 {
6039     \IfPackageLoadedTF { tikz }
6040     {
6041         \IfPackageLoadedTF { booktabs }
6042         { #2 }
6043         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6044     }
6045     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6046 }
6047 \NewExpandableDocumentCommand { \@@_TopRule } { }
6048 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6049 \cs_new:Npn \@@_TopRule_i:
6050 {
6051     \noalign \bgroup
6052     \peek_meaning:NTF [
6053     { \@@_TopRule_ii: }
6054     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6055 }
6056 \NewDocumentCommand \@@_TopRule_ii: { o }
6057 {
6058     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6059     {
6060         \@@_hline:n
6061         {
6062             position = \int_eval:n { \c@iRow + 1 } ,
6063             tikz =
6064             {
6065                 line~width = #1 ,
6066                 yshift = 0.25 \arrayrulewidth ,

```

```

6067             shorten~< = - 0.5 \arrayrulewidth
6068         } ,
6069         total-width = #1
6070     }
6071 }
6072 \skip_vertical:n { \belowrulesep + #1 }
6073 \egroup
6074 }

6075 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6076 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6077 \cs_new:Npn \@@_BottomRule_i:
6078 {
6079     \noalign \bgroup
6080     \peek_meaning:NTF [
6081         { \@@_BottomRule_ii: }
6082         { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6083     }

6084 \NewDocumentCommand \@@_BottomRule_ii: { o }
6085 {
6086     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6087     {
6088         \@@_hline:n
6089         {
6090             position = \int_eval:n { \c@iRow + 1 } ,
6091             tikz =
6092             {
6093                 line-width = #1 ,
6094                 yshift = 0.25 \arrayrulewidth ,
6095                 shorten~< = - 0.5 \arrayrulewidth
6096             } ,
6097             total-width = #1 ,
6098         }
6099     }
6100     \skip_vertical:N \aboverulesep
6101     \@@_create_row_node_i:
6102     \skip_vertical:n { #1 }
6103     \egroup
6104 }

6105 \NewExpandableDocumentCommand { \@@_MidRule } { }
6106 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6107 \cs_new:Npn \@@_MidRule_i:
6108 {
6109     \noalign \bgroup
6110     \peek_meaning:NTF [
6111         { \@@_MidRule_ii: }
6112         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6113     }

6114 \NewDocumentCommand \@@_MidRule_ii: { o }
6115 {
6116     \skip_vertical:N \aboverulesep
6117     \@@_create_row_node_i:
6118     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6119     {
6120         \@@_hline:n
6121         {
6122             position = \int_eval:n { \c@iRow + 1 } ,
6123             tikz =
6124             {
6125                 line-width = #1 ,
6126                 yshift = 0.25 \arrayrulewidth ,
6127                 shorten~< = - 0.5 \arrayrulewidth

```

```

6128         } ,
6129         total-width = #1 ,
6130     }
6131 }
6132 \skip_vertical:n { \belowrulesep + #1 }
6133 \egroup
6134 }

```

## General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline: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. However, unknown keys will be analyzed further with another set of keys.

```

6135 \keys_define:nn { nicematrix / Rules }
6136 {
6137     position .int_set:N = \l_@@_position_int ,
6138     position .value_required:n = true ,
6139     start .int_set:N = \l_@@_start_int ,
6140     end .code:n =
6141         \bool_lazy_or:nnTF
6142         { \tl_if_empty_p:n { #1 } }
6143         { \str_if_eq_p:ee { #1 } { last } }
6144         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6145         { \int_set:Nn \l_@@_end_int { #1 } }
6146 }

```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6147 \keys_define:nn { nicematrix / RulesBis }
6148 {
6149     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6150     multiplicity .initial:n = 1 ,
6151     dotted .bool_set:N = \l_@@_dotted_bool ,
6152     dotted .initial:n = false ,
6153     dotted .default: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]`.

```

6154     color .code:n =
6155         \@@_set_CTarc:n { #1 }
6156         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6157     color .value_required:n = true ,
6158     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6159     sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```

6160     tikz .code:n =
6161         \IfPackageLoadedTF { tikz }
6162         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6163         { \@@_error:n { tikz-without-tikz } } ,
6164     tikz .value_required:n = true ,
6165     total-width .dim_set:N = \l_@@_rule_width_dim ,

```

```

6166     total-width .value_required:n = true ,
6167     width .meta:n = { total-width = #1 } ,
6168     unknown .code:n = \@_error:n { Unknown~key~for~RulesBis }
6169 }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6170 \cs_new_protected:Npn \@_vline:n #1
6171 {

```

The group is for the options.

```

6172     \group_begin:
6173     \int_set_eq:NN \l_@@_end_int \c@iRow
6174     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

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).

```

6175     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6176     \@_vline_i:
6177     \group_end:
6178 }

```

```

6179 \cs_new_protected:Npn \@_vline_i:
6180 {

```

`\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`.

```

6181     \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6182     \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6183     \l_tmpa_tl
6184     {

```

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 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.

```

6185         \bool_gset_true:N \g_tmpa_bool
6186         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6187         { \@_test_vline_in_block:nnnn ##1 }
6188         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6189         { \@_test_vline_in_block:nnnn ##1 }
6190         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6191         { \@_test_vline_in_stroken_block:nnnn ##1 }
6192         \clist_if_empty:NF \l_@@_corners_clist { \@_test_in_corner_v: }
6193         \bool_if:NTF \g_tmpa_bool
6194         {
6195             \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6196             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6197         }
6198     {
6199         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6200         {
6201             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6202             \@_vline_ii:
6203             \int_zero:N \l_@@_local_start_int
6204         }
6205     }
6206 }
6207 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }

```

```

6208     {
6209         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6210         \@@_vline_ii:
6211     }
6212 }

6213 \cs_new_protected:Npn \@@_test_in_corner_v:
6214 {
6215     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6216     {
6217         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6218         { \bool_set_false:N \g_tmpa_bool }
6219     }
6220     {
6221         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6222         {
6223             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6224             { \bool_set_false:N \g_tmpa_bool }
6225             {
6226                 \@@_if_in_corner:nT
6227                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6228                 { \bool_set_false:N \g_tmpa_bool }
6229             }
6230         }
6231     }
6232 }

6233 \cs_new_protected:Npn \@@_vline_ii:
6234 {
6235     \tl_clear:N \l_@@_tikz_rule_tl
6236     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6237     \bool_if:NNTF \l_@@_dotted_bool
6238     { \@@_vline_iv: }
6239     {
6240         \tl_if_empty:NNTF \l_@@_tikz_rule_tl
6241         { \@@_vline_iii: }
6242         { \@@_vline_v: }
6243     }
6244 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6245 \cs_new_protected:Npn \@@_vline_iii:
6246 {
6247     \pgfpicture
6248     \pgfrememberpicturepositiononpagetrue
6249     \pgf@relevantforpicturesizefalse
6250     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6251     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6252     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6253     \dim_set:Nn \l_tmpb_dim
6254     {
6255         \pgf@x
6256         - 0.5 \l_@@_rule_width_dim
6257         +
6258         ( \arrayrulewidth * \l_@@_multiplicity_int
6259           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6260     }
6261     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6262     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6263     \bool_lazy_all:nT
6264     {

```

```

6265     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6266     { \cs_if_exist_p:N \CT@drsc@ }
6267     { ! \tl_if_blank_p:o \CT@drsc@ }
6268   }
6269   {
6270     \group_begin:
6271     \CT@drsc@
6272     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6273     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6274     \dim_set:Nn \l_@@_tmpd_dim
6275       {
6276         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6277         * ( \l_@@_multiplicity_int - 1 )
6278       }
6279     \pgfpathrectanglecorners
6280       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6281       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6282     \pgfusepath { fill }
6283     \group_end:
6284   }
6285   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6286   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6287   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6288   {
6289     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6290     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6291     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6292     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6293   }
6294   \CT@arc@
6295   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6296   \pgfsetrectcap
6297   \pgfusepathqstroke
6298   \endpgfpicture
6299 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6300 \cs_new_protected:Npn \@@_vline_iv:
6301 {
6302   \pgfpicture
6303   \pgfrememberpicturepositiononpagetrue
6304   \pgf@relevantforpicturesizefalse
6305   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6306   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6307   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6308   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6309   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6310   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6311   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6312   \CT@arc@
6313   \@@_draw_line:
6314   \endpgfpicture
6315 }

```

The following code is for the case when the user uses the key `tikz`.

```

6316 \cs_new_protected:Npn \@@_vline_v:
6317 {
6318   \begin { tikzpicture }

```

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).

```

6319 \CT@arc@
6320 \tl_if_empty:NF \l_@@_rule_color_tl
6321 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6322 \pgfrememberpicturepositiononpagetrue
6323 \pgf@relevantforpicturesizefalse
6324 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6325 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6326 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6327 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6328 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6329 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6330 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6331 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6332 ( \l_tmpb_dim , \l_tmpa_dim ) --
6333 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6334 \end { tikzpicture }
6335 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

6336 \cs_new_protected:Npn \@@_draw_vlines:
6337 {
6338   \int_step_inline:nnn
6339   {
6340     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6341     { 2 }
6342     { 1 }
6343   }
6344   {
6345     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6346     { \c@jCol }
6347     { \int_eval:n { \c@jCol + 1 } }
6348   }
6349   {
6350     \str_if_eq:eeF \l_@@_vlines_clist { all }
6351     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6352     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6353   }
6354 }

```

## The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{nicematrix/Rules}`.

```

6355 \cs_new_protected:Npn \@@_hline:n #1
6356 {

```

The group is for the options.

```

6357   \group_begin:
6358   \int_set_eq:NN \l_@@_end_int \c@jCol
6359   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6360   \@@_hline_i:
6361   \group_end:
6362 }

6363 \cs_new_protected:Npn \@@_hline_i:
6364 {
6365   % \int_zero:N \l_@@_local_start_int
6366   % \int_zero:N \l_@@_local_end_int

```

`\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`.

```

6367 \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6368 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6369 \l_tmpb_tl
6370 {

```

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 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.

```

6371 \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6372 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6373 { \@@_test_hline_in_block:nnnnn ##1 }

6374 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6375 { \@@_test_hline_in_block:nnnnn ##1 }

6376 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6377 { \@@_test_hline_in_stroken_block:nnnn ##1 }

6378 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6379 \bool_if:NTF \g_tmpa_bool
6380 {
6381 \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6382 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6383 }
6384 {
6385 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6386 {
6387 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6388 \@@_hline_ii:
6389 \int_zero:N \l_@@_local_start_int
6390 }
6391 }
6392 }
6393 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6394 {
6395 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6396 \@@_hline_ii:
6397 }
6398 }

```

```

6399 \cs_new_protected:Npn \@@_test_in_corner_h:
6400 {
6401 \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6402 {
6403 \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6404 { \bool_set_false:N \g_tmpa_bool }
6405 }
6406 {
6407 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6408 {
6409 \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6410 { \bool_set_false:N \g_tmpa_bool }
6411 {
6412 \@@_if_in_corner:nT
6413 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6414 { \bool_set_false:N \g_tmpa_bool }
6415 }
6416 }
6417 }
6418 }

```



```

6419 \cs_new_protected:Npn \l_@@_hline_ii:
6420 {
6421   \tl_clear:N \l_@@_tikz_rule_tl
6422   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6423   \bool_if:NTF \l_@@_dotted_bool
6424     { \l_@@_hline_iv: }
6425     {
6426       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6427         { \l_@@_hline_iii: }
6428         { \l_@@_hline_v: }
6429     }
6430 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6431 \cs_new_protected:Npn \l_@@_hline_iii:
6432 {
6433   \pgfpicture
6434   \pgfrememberpicturepositiononpagetrue
6435   \pgf@relevantforpicturesizefalse
6436   \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6437   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6438   \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6439   \dim_set:Nn \l_tmpb_dim
6440     {
6441       \pgf@y
6442       - 0.5 \l_@@_rule_width_dim
6443       +
6444       ( \arrayrulewidth * \l_@@_multiplicity_int
6445         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6446     }
6447   \l_@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6448   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6449   \bool_lazy_all:nT
6450     {
6451       { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6452       { \cs_if_exist_p:N \CT@drsc@ }
6453       { ! \tl_if_blank_p:o \CT@drsc@ }
6454     }
6455     {
6456       \group_begin:
6457       \CT@drsc@
6458       \dim_set:Nn \l_@@_tmpd_dim
6459         {
6460           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6461           * ( \l_@@_multiplicity_int - 1 )
6462         }
6463       \pgfpathrectanglecorners
6464         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6465         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6466       \pgfusepathqfill
6467       \group_end:
6468     }
6469   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6470   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6471   \prg_replicate:nm { \l_@@_multiplicity_int - 1 }
6472     {
6473       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6474       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6475       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6476       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6477     }
6478   \CT@arc@
6479   \pgfsetlinewidth { 1.1 \arrayrulewidth }

```

```

6480 \pgfsetrectcap
6481 \pgfusepathqstroke
6482 \endpgfpicture
6483 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6484 \cs_new_protected:Npn \@@_hline_iv:
6485 {
6486   \pgfpicture
6487   \pgfrememberpicturepositiononpagetrue
6488   \pgf@relevantforpicturesizefalse
6489   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6490   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6491   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6492   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6493   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6494   \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6495   {
6496     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6497     \bool_if:NF \g_@@_delims_bool
6498     { \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.

```

6499   \tl_if_eq:NnF \g_@@_left_delim_tl (
6500     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6501   )
6502   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6503   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6504   \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }
6505   {
6506     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6507     \bool_if:NF \g_@@_delims_bool
6508     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6509     \tl_if_eq:NnF \g_@@_right_delim_tl )
6510     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6511   }
6512   \CT@arc@
6513   \@@_draw_line:
6514   \endpgfpicture
6515 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

6516 \cs_new_protected:Npn \@@_hline_v:

```

```

6517 {
6518   \begin { tikzpicture }

```

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).

```

6519   \CT@arc@
6520   \tl_if_empty:NF \l_@@_rule_color_tl
6521     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6522   \pgfrememberpicturepositiononpagetrue
6523   \pgf@relevantforpicturesizefalse
6524   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6525   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6526   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6527   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6528   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6529   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6530   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6531   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6532     ( \l_tmpa_dim , \l_tmpb_dim ) --
6533     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6534   \end { tikzpicture }
6535 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners — if the key `corners` is used).

```

6536 \cs_new_protected:Npn \@@_draw_hlines:
6537 {
6538   \int_step_inline:nnn
6539     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6540     {
6541       \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6542       { \c@iRow }
6543       { \int_eval:n { \c@iRow + 1 } } }
6544   }
6545   {
6546     \str_if_eq:eeF \l_@@_hlines_clist { all }
6547     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6548     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6549   }
6550 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6551 \cs_set:Npn \@@_Hline: { \noalign \bgroup \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

6552 \cs_set:Npn \@@_Hline_i:n #1
6553 {
6554   \peek_remove_spaces:n
6555   {
6556     \peek_meaning:NTF \Hline
6557     { \@@_Hline_ii:nn { #1 + 1 } }
6558     { \@@_Hline_iii:n { #1 } }
6559   }
6560 }
6561 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6562 \cs_set:Npn \@@_Hline_iii:n #1
6563 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }

```

```

6564 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6565 {
6566   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6567   \skip_vertical:N \l_@@_rule_width_dim
6568   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6569   {
6570     \@@_hline:n
6571     {
6572       multiplicity = #1 ,
6573       position = \int_eval:n { \c@iRow + 1 } ,
6574       total-width = \dim_use:N \l_@@_rule_width_dim ,
6575       #2
6576     }
6577   }
6578   \egroup
6579 }

```

### 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`).

```

6580 \cs_new_protected:Npn \@@_custom_line:n #1
6581 {
6582   \str_clear_new:N \l_@@_command_str
6583   \str_clear_new:N \l_@@_ccommand_str
6584   \str_clear_new:N \l_@@_letter_str
6585   \tl_clear_new:N \l_@@_other_keys_tl
6586   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

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.

```

6587   \bool_lazy_all:nTF
6588   {
6589     { \str_if_empty_p:N \l_@@_letter_str }
6590     { \str_if_empty_p:N \l_@@_command_str }
6591     { \str_if_empty_p:N \l_@@_ccommand_str }
6592   }
6593   { \@@_error:n { No~letter~and~no~command } }
6594   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6595 }
6596 \keys_define:nn { nicematrix / custom-line }
6597 {
6598   letter .str_set:N = \l_@@_letter_str ,
6599   letter .value_required:n = true ,
6600   command .str_set:N = \l_@@_command_str ,
6601   command .value_required:n = true ,
6602   ccommand .str_set:N = \l_@@_ccommand_str ,
6603   ccommand .value_required:n = true ,
6604 }

```

```

6605 \cs_new_protected:Npn \@@_custom_line_i:n #1
6606 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6607   \bool_set_false:N \l_@@_tikz_rule_bool
6608   \bool_set_false:N \l_@@_dotted_rule_bool
6609   \bool_set_false:N \l_@@_color_bool

```

```

6610 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6611 \bool_if:NT \l_@@_tikz_rule_bool
6612 {
6613   \IfPackageLoadedF { tikz }
6614   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6615   \bool_if:NT \l_@@_color_bool
6616   { \@@_error:n { color-in-custom-line-with-tikz } }
6617 }
6618 \bool_if:NT \l_@@_dotted_rule_bool
6619 {
6620   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6621   { \@@_error:n { key-multiplicity-with-dotted } }
6622 }
6623 \str_if_empty:NF \l_@@_letter_str
6624 {
6625   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6626   { \@@_error:n { Several-letters } }
6627   {
6628     \tl_if_in:NoTF
6629     \c_@@_forbidden_letters_str
6630     \l_@@_letter_str
6631     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6632   }

```

During the analyse 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.

```

6633 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6634 { \@@_v_custom_line:n { #1 } }
6635 }
6636 }
6637 }
6638 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6639 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6640 }
6641 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6642 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6643 \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}`). That's why the following set of keys has some keys which are no-op.

```

6644 \keys_define:nn { nicematrix / custom-line-bis }
6645 {
6646   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6647   multiplicity .initial:n = 1 ,
6648   multiplicity .value_required:n = true ,
6649   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6650   color .value_required:n = true ,
6651   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6652   tikz .value_required:n = true ,
6653   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6654   dotted .value_forbidden:n = true ,
6655   total-width .code:n = { } ,
6656   total-width .value_required:n = true ,
6657   width .code:n = { } ,
6658   width .value_required:n = true ,
6659   sep-color .code:n = { } ,
6660   sep-color .value_required:n = true ,
6661   unknown .code:n = \@@_error:n { Unknown-key-for-custom-line }
6662 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
6663 \bool_new:N \l_@@_dotted_rule_bool
6664 \bool_new:N \l_@@_tikz_rule_bool
6665 \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). The key `width` is deprecated and has been replaced by the key `total-width`.

```
6666 \keys_define:nn { nicematrix / custom-line-width }
6667 {
6668   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6669   multiplicity .initial:n = 1 ,
6670   multiplicity .value_required:n = true ,
6671   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6672   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6673   \bool_set_true:N \l_@@_total_width_bool ,
6674   total-width .value_required:n = true ,
6675   width .meta:n = { total-width = #1 } ,
6676   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6677 }
```

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 `\@@_hline:n` (which is in the internal `\CodeAfter`).

```
6678 \cs_new_protected:Npn \@@_h_custom_line:n #1
6679 {
```

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`).

```
6680   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6681   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6682 }
```

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 `\@@_hline:n` (which is in the internal `\CodeAfter`).

```
6683 \cs_new_protected:Npn \@@_c_custom_line:n #1
6684 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6685   \exp_args:Nc \NewExpandableDocumentCommand
6686   { nicematrix - \l_@@_ccommand_str }
6687   { 0 { } m }
6688   {
6689     \noalign
6690     {
6691       \@@_compute_rule_width:n { #1 , ##1 }
6692       \skip_vertical:n { \l_@@_rule_width_dim }
6693       \clist_map_inline:nn
6694       { ##2 }
6695       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6696     }
6697   }
6698   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6699 }
```

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*.

```

6700 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6701 {
6702   \tl_if_in:nnTF { #2 } { - }
6703   { \@@_cut_on_hyphen:w #2 \q_stop }
6704   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6705   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6706   {
6707     \@@_hline:n
6708     {
6709       #1 ,
6710       start = \l_tmpa_tl ,
6711       end = \l_tmpb_tl ,
6712       position = \int_eval:n { \c@iRow + 1 } ,
6713       total-width = \dim_use:N \l_@@_rule_width_dim
6714     }
6715   }
6716 }

6717 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6718 {
6719   \bool_set_false:N \l_@@_tikz_rule_bool
6720   \bool_set_false:N \l_@@_total_width_bool
6721   \bool_set_false:N \l_@@_dotted_rule_bool
6722   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6723   \bool_if:NF \l_@@_total_width_bool
6724   {
6725     \bool_if:NTF \l_@@_dotted_rule_bool
6726     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6727     {
6728       \bool_if:NF \l_@@_tikz_rule_bool
6729       {
6730         \dim_set:Nn \l_@@_rule_width_dim
6731         {
6732           \arrayrulewidth * \l_@@_multiplicity_int
6733           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6734         }
6735       }
6736     }
6737   }
6738 }

6739 \cs_new_protected:Npn \@@_v_custom_line:n #1
6740 {
6741   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6742   \tl_gput_right:Ne \g_@@_array_preamble_tl
6743   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6744   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6745   {
6746     \@@_vline:n
6747     {
6748       #1 ,
6749       position = \int_eval:n { \c@jCol + 1 } ,
6750       total-width = \dim_use:N \l_@@_rule_width_dim
6751     }
6752   }
6753   \@@_rec_preamble:n
6754 }

6755 \@@_custom_line:n
6756 { letter = : , command = hdottedline , ccommand = cdottedline , dotted }

```

## 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 `\l_tmpa_bool` is set to false.

```

6757 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6758 {
6759   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6760   {
6761     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6762     {
6763       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6764       {
6765         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6766         { \bool_gset_false:N \g_tmpa_bool }
6767       }
6768     }
6769   }
6770 }

```

The same for vertical rules.

```

6771 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6772 {
6773   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6774   {
6775     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6776     {
6777       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6778       {
6779         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6780         { \bool_gset_false:N \g_tmpa_bool }
6781       }
6782     }
6783   }
6784 }

6785 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6786 {
6787   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6788   {
6789     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6790     {
6791       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6792       { \bool_gset_false:N \g_tmpa_bool }
6793       {
6794         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6795         { \bool_gset_false:N \g_tmpa_bool }
6796       }
6797     }
6798   }
6799 }

6800 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6801 {
6802   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6803   {
6804     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6805     {
6806       \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6807       { \bool_gset_false:N \g_tmpa_bool }
6808       {
6809         \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6810         { \bool_gset_false:N \g_tmpa_bool }
6811       }
6812     }
6813   }
6814 }

```



```

6812     }
6813   }
6814 }

```

## 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.

```

6815 \cs_new_protected:Npn \@@_compute_corners:
6816 {
6817   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6818     { \@@_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`).

```

6819   \clist_clear:N \l_@@_corners_cells_clist
6820   \clist_map_inline:Nn \l_@@_corners_clist
6821     {
6822       \str_case:nnF { ##1 }
6823         {
6824           { NW }
6825           { \@@_compute_a_corner:nnnnnn 1 1 1 \c@iRow \c@jCol }
6826           { NE }
6827           { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6828           { SW }
6829           { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6830           { SE }
6831           { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6832         }
6833       { \@@_error:nn { bad~corner } { ##1 } }
6834     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6835   \clist_if_empty:NF \l_@@_corners_cells_clist
6836   {

```

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.

```

6837     \tl_gput_right:Ne \g_@@_aux_tl
6838     {
6839       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6840         { \l_@@_corners_cells_clist }
6841     }
6842   }
6843 }

```

```

6844 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6845 {
6846   \int_step_inline:nnn { #1 } { #3 }
6847   {
6848     \int_step_inline:nnn { #2 } { #4 }
6849     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6850   }
6851 }

```

```

6852 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6853 {
6854   \cs_if_exist:cTF
6855     { @@_block_ \int_eval:n { #1 } - \int_eval:n { #2 } }
6856     { \prg_return_true: }
6857     { \prg_return_false: }
6858 }

```

“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_a_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.

```

6859 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6860 {

```

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.

```

6861   \bool_set_false:N \l_tmpa_bool
6862   \int_zero_new:N \l_@@_last_empty_row_int
6863   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6864   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6865   {
6866     \bool_lazy_or:nnTF
6867     {
6868       \cs_if_exist_p:c
6869       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6870     }
6871     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6872     { \bool_set_true:N \l_tmpa_bool }
6873     {
6874       \bool_if:NF \l_tmpa_bool
6875       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6876     }
6877   }

```

Now, you determine the last empty cell in the row of number 1.

```

6878   \bool_set_false:N \l_tmpa_bool
6879   \int_zero_new:N \l_@@_last_empty_column_int
6880   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6881   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6882   {
6883     \bool_lazy_or:nnTF
6884     {
6885       \cs_if_exist_p:c
6886       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6887     }
6888     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6889     { \bool_set_true:N \l_tmpa_bool }
6890     {
6891       \bool_if:NF \l_tmpa_bool
6892       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6893     }
6894   }

```

Now, we loop over the rows.

```
6895 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6896 {
```

We treat the row number ##1 with another loop.

```
6897 \bool_set_false:N \l_tmpa_bool
6898 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6899 {
6900 \bool_lazy_or:nnTF
6901 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6902 { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6903 { \bool_set_true:N \l_tmpa_bool }
6904 {
6905 \bool_if:NF \l_tmpa_bool
6906 {
6907 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6908 \clist_put_right:Nn
6909 \l_@@_corners_cells_clist
6910 { ##1 - #####1 }
6911 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6912 }
6913 }
6914 }
6915 }
6916 }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```
6917 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6918 \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:NnT \l_@@_corners_cells_clist { #1 } ...`

## 24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
6919 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6920 \keys_define:nn { nicematrix / NiceMatrixBlock }
6921 {
6922 auto-columns-width .code:n =
6923 {
6924 \bool_set_true:N \l_@@_block_auto_columns_width_bool
6925 \dim_gzero_new:N \g_@@_max_cell_width_dim
6926 \bool_set_true:N \l_@@_auto_columns_width_bool
6927 }
6928 }
```

```
6929 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6930 {
6931 \int_gincr:N \g_@@_NiceMatrixBlock_int
6932 \dim_zero:N \l_@@_columns_width_dim
6933 \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6934 \bool_if:NT \l_@@_block_auto_columns_width_bool
6935 {
6936 \cs_if_exist:cT
```

```

6937     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6938     {
6939         \dim_set:Nn \l_@@_columns_width_dim
6940         {
6941             \use:c
6942             { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6943         }
6944     }
6945 }
6946 }

```

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`).

```

6947 {
6948     \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.

```

6949     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6950     {
6951         \bool_if:NT \l_@@_block_auto_columns_width_bool
6952         {
6953             \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6954             \iow_shipout:Ne \@mainaux
6955             {
6956                 \cs_gset:cpn
6957                 { @@ _ 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.

```

6958             { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6959         }
6960         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6961     }
6962 }
6963 \ignorespacesafterend
6964 }

```

## 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).

```

6965 \cs_new_protected:Npn \@@_create_extra_nodes:
6966 {
6967     \bool_if:nTF \l_@@_medium_nodes_bool
6968     {
6969         \bool_if:NTF \l_@@_no_cell_nodes_bool
6970         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6971         {
6972             \bool_if:NTF \l_@@_large_nodes_bool
6973             \@@_create_medium_and_large_nodes:
6974             \@@_create_medium_nodes:
6975         }
6976     }
6977     {
6978         \bool_if:NT \l_@@_large_nodes_bool
6979         {
6980             \bool_if:NTF \l_@@_no_cell_nodes_bool

```

```

6981         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6982         \@@_create_large_nodes:
6983     }
6984 }
6985 }

```

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`.

```

6986 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6987 {
6988     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6989     {
6990         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6991         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6992         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6993         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
6994     }
6995     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6996     {
6997         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
6998         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
6999         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7000         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7001     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7002     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7003     {
7004         \int_step_variable:nnNn
7005         \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.

```

7006     {
7007         \cs_if_exist:cT
7008         { 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`.

```

7009     {
7010         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7011         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7012         { \dim_min:vn { l_@@_row_ \@@_i: _min_dim } \pgf@y }
7013         \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7014         {
7015             \dim_set:cn { l_@@_column_ \@@_j: _min_dim }
7016             { \dim_min:vn { l_@@_column_ \@@_j: _min_dim } \pgf@x }
7017         }

```

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`.

```

7018     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7019     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7020     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } { \pgf@y } }
7021     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7022     {
7023         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7024         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } { \pgf@x } }
7025     }
7026 }
7027 }
7028 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7029 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7030 {
7031     \dim_compare:nNnT
7032     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7033     {
7034         \@@_qpoint:n { row - \@@_i: - base }
7035         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7036         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7037     }
7038 }
7039 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7040 {
7041     \dim_compare:nNnT
7042     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7043     {
7044         \@@_qpoint:n { col - \@@_j: }
7045         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7046         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7047     }
7048 }
7049 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7050 \cs_new_protected:Npn \@@_create_medium_nodes:
7051 {
7052     \pgfpicture
7053     \pgfrememberpicturepositiononpagetrue
7054     \pgf@relevantforpicturesizefalse
7055     \@@_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”.

```

7056     \tl_set:Nn \l_@@_suffix_tl { -medium }
7057     \@@_create_nodes:
7058     \endpgfpicture
7059 }

```

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:`.

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7060 \cs_new_protected:Npn \@@_create_large_nodes:
7061 {
7062   \pgfpicture
7063     \pgfrememberpicturepositiononpagetrue
7064     \pgf@relevantforpicturesizefalse
7065     \@@_computations_for_medium_nodes:
7066     \@@_computations_for_large_nodes:
7067     \tl_set:Nn \l_@@_suffix_tl { - large }
7068     \@@_create_nodes:
7069   \endpgfpicture
7070 }
7071 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7072 {
7073   \pgfpicture
7074     \pgfrememberpicturepositiononpagetrue
7075     \pgf@relevantforpicturesizefalse
7076     \@@_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”.

```

7077     \tl_set:Nn \l_@@_suffix_tl { - medium }
7078     \@@_create_nodes:
7079     \@@_computations_for_large_nodes:
7080     \tl_set:Nn \l_@@_suffix_tl { - large }
7081     \@@_create_nodes:
7082   \endpgfpicture
7083 }

```

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.

```

7084 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7085 {
7086   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7087   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

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`.

```

7088   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7089   {
7090     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7091     {
7092       (
7093         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7094         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7095       )
7096       / 2
7097     }
7098     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7099     { l_@@_row _ \@@_i: _ min _ dim }
7100   }
7101   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7102   {
7103     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7104     {
7105       (
7106         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7107         \dim_use:c
7108           { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7109       )
7110       / 2
7111     }
7112     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7113     { l_@@_column _ \@@_j: _ max _ dim }
7114   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7115 \dim_sub:cn
7116 { l_@@_column _ 1 _ min _ dim }
7117 \l_@@_left_margin_dim
7118 \dim_add:cn
7119 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7120 \l_@@_right_margin_dim
7121 }

```

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).

```

7122 \cs_new_protected:Npn \@@_create_nodes:
7123 {
7124   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7125   {
7126     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7127     {

```

We draw the rectangular node for the cell (`\@@_i`-`\@@_j`).

```

7128       \@@_pgf_rect_node:nnnnn
7129       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7130       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7131       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7132       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7133       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7134       \str_if_empty:NF \l_@@_name_str
7135       {
7136         \pgfnodealias
7137         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7138         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7139       }
7140     }
7141   }
7142   \int_step_inline:nn { \c@iRow }
7143   {
7144     \pgfnodealias
7145     { \@@_env: - ##1 - last \l_@@_suffix_tl }
7146     { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7147   }
7148   \int_step_inline:nn { \c@jCol }
7149   {
7150     \pgfnodealias
7151     { \@@_env: - last - ##1 \l_@@_suffix_tl }
7152     { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7153   }
7154   \pgfnodealias % added 2025-04-05
7155   { \@@_env: - last - last \l_@@_suffix_tl }
7156   { \@@_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  $n$ .

```

7157   \seq_map_pairwise_function:NNN
7158   \g_@@_multicolumn_cells_seq
7159   \g_@@_multicolumn_sizes_seq
7160   \@@_node_for_multicolumn:nn
7161 }

```



```

7162 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7163 {
7164   \cs_set_nopar:Npn \@@_i: { #1 }
7165   \cs_set_nopar:Npn \@@_j: { #2 }
7166 }

```

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”).

```

7167 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7168 {
7169   \@@_extract_coords_values: #1 \q_stop
7170   \@@_pgf_rect_node:nnnnn
7171   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7172   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7173   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7174   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7175   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7176   \str_if_empty:NF \l_@@_name_str
7177   {
7178     \pgfnodealias
7179     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7180     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7181   }
7182 }

```

## 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).

```

7183 \keys_define:nn { nicematrix / Block / FirstPass }
7184 {
7185   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7186   \bool_set_true:N \l_@@_p_block_bool ,
7187   j .value_forbidden:n = true ,
7188   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7189   l .value_forbidden:n = true ,
7190   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7191   r .value_forbidden:n = true ,
7192   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7193   c .value_forbidden:n = true ,
7194   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7195   L .value_forbidden:n = true ,
7196   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7197   R .value_forbidden:n = true ,
7198   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7199   C .value_forbidden:n = true ,
7200   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7201   t .value_forbidden:n = true ,
7202   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7203   T .value_forbidden:n = true ,
7204   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7205   b .value_forbidden:n = true ,
7206   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7207   B .value_forbidden:n = true ,

```

```

7208 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7209 m .value_forbidden:n = true ,
7210 v-center .meta:n = m ,
7211 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7212 p .value_forbidden:n = true ,
7213 color .code:n =
7214   \@@_color:n { #1 }
7215   \tl_set_rescan:Nnn
7216     \l_@@_draw_tl
7217     { \char_set_catcode_other:N ! }
7218     { #1 } ,
7219 color .value_required:n = true ,
7220 respect-arraystretch .code:n =
7221   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7222 respect-arraystretch .value_forbidden:n = true ,
7223 }

```

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.

```

7224 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7225 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7226 {

```

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).

```

7227   \tl_if_blank:nTF { #2 }
7228   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7229   {
7230     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7231     \@@_Block_i_czech:w \@@_Block_i:w
7232     #2 \q_stop
7233   }
7234   { #1 } { #3 } { #4 }
7235   \ignorespaces
7236 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7237 \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`.

```

7238 {
7239   \char_set_catcode_active:N -
7240   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7241 }

```

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.

```

7242 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7243 {

```

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).

```

7244   \bool_lazy_or:nnTF

```

```

7245 { \tl_if_blank_p:n { #1 } }
7246 { \str_if_eq_p:ee { * } { #1 } }
7247 { \int_set:Nn \l_tmpa_int { 100 } }
7248 { \int_set:Nn \l_tmpa_int { #1 } }
7249 \bool_lazy_or:nnTF
7250 { \tl_if_blank_p:n { #2 } }
7251 { \str_if_eq_p:ee { * } { #2 } }
7252 { \int_set:Nn \l_tmpb_int { 100 } }
7253 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7254 \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7255 {
7256   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7257   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7258   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7259 }
7260 { \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.

```

7261 \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7262 \tl_set:Nx \l_tmpa_tl
7263 {
7264   { \int_use:N \c@iRow }
7265   { \int_use:N \c@jCol }
7266   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7267   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7268 }

```

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).

```

7269 \bool_set_false:N \l_tmpa_bool
7270 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7271 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7272 \bool_case:nF
7273 {
7274   \l_tmpa_bool { \@@_Block_vii:eennn }
7275   \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right now 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.

```

7276 \l_@@_X_bool { \@@_Block_v:eennn }
7277 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7278 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7279 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7280 }
7281 { \@@_Block_v:eennn }
7282 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7283 }

```

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 now 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.

```

7284 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7285 {
7286   \int_gincr:N \g_@@_block_box_int
7287   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7288   {
7289     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7290     {
7291       \@@_actually_diagbox:nnnnnn
7292       { \int_use:N \c@iRow }
7293       { \int_use:N \c@jCol }
7294       { \int_eval:n { \c@iRow + #1 - 1 } }
7295       { \int_eval:n { \c@jCol + #2 - 1 } }
7296       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7297       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7298     }
7299   }
7300   \box_gclear_new:c
7301   { \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!

```

7302   \hbox_gset:cn
7303   { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7304   {

```

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`).

```

7305   \tl_if_empty:NTF \l_@@_color_tl
7306   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7307   { \@@_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`.

```

7308   \int_compare:nNnT { #1 } = { \c_one_int }
7309   {
7310     \int_if_zero:nTF { \c@iRow }
7311     {

```

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}$

```

```

7312         \cs_set_eq:NN \Block \@@_NullBlock:
7313         \l_@@_code_for_first_row_tl
7314     }
7315     {
7316         \int_compare:nNnT { \c{iRow} } = { \l_@@_last_row_int }
7317         {
7318             \cs_set_eq:NN \Block \@@_NullBlock:
7319             \l_@@_code_for_last_row_tl
7320         }
7321     }
7322     \g_@@_row_style_tl
7323 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7324     \@@_reset_arraystretch:
7325     \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7326     #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.).

```

7327     \@@_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}`.

```

7328     \bool_if:NTF \l_@@_tabular_bool
7329     {
7330         \bool_lazy_all:nTF
7331         {
7332             { \int_compare_p:nNn { #2 } = { \c_one_int } }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of  $-1$  cm.

```

7333     {
7334         ! \dim_compare_p:nNn
7335         { \l_@@_col_width_dim } < { \c_zero_dim }
7336     }
7337     { ! \g_@@_rotate_bool }
7338 }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7339     {
7340         \use:e
7341         {

```

The `\exp_not:N` is mandatory before `\begin`. It will be possible to delete the `\exp_not:N` in TeXLive 2025 because `\begin` is now protected by `\protected` (and not by `\protect`). There is several other occurrences in that document.

```

7342         \exp_not:N \begin { minipage }
7343         [ \str_lowercase:f \l_@@_vpos_block_str ]

```

```

7344         { \l_@@_col_width_dim }
7345         \str_case:on \l_@@_hpos_block_str
7346         { c \centering r \raggedleft l \raggedright }
7347     }
7348     #5
7349     \end { minipage }
7350 }

```

In the other cases, we use a `{tabular}`.

```

7351 {
7352     \bool_if:NT \c_@@_testphase_table_bool
7353     { \tagpdfsetup { table / tagging = presentation } }
7354     \use:e
7355     {
7356         \exp_not:N \begin { tabular }
7357         [ \str_lowercase:f \l_@@_vpos_block_str ]
7358         { @ { } \l_@@_hpos_block_str @ { } }
7359     }
7360     #5
7361     \end { tabular }
7362 }
7363 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7364 {
7365     \c_math_toggle_token
7366     \use:e
7367     {
7368         \exp_not:N \begin { array }
7369         [ \str_lowercase:f \l_@@_vpos_block_str ]
7370         { @ { } \l_@@_hpos_block_str @ { } }
7371     }
7372     #5
7373     \end { array }
7374     \c_math_toggle_token
7375 }
7376 }

```

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).

```

7377     \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.

```

7378     \int_compare:nNnT { #2 } = { \c_one_int }
7379     {
7380         \dim_gset:Nn \g_@@_blocks_wd_dim
7381         {
7382             \dim_max:nn
7383             { \g_@@_blocks_wd_dim }
7384             {
7385                 \box_wd:c
7386                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7387             }
7388         }
7389     }

```

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.

```

7390     \bool_lazy_and:nnT
7391     { \int_compare_p:nNn { #1 } = { \c_one_int } }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7392     { \str_if_empty_p:N \l_@@_vpos_block_str }
7393     {
7394         \dim_gset:Nn \g_@@_blocks_ht_dim
7395         {
7396             \dim_max:nn
7397             { \g_@@_blocks_ht_dim }
7398             {
7399                 \box_ht:c
7400                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7401             }
7402         }
7403         \dim_gset:Nn \g_@@_blocks_dp_dim
7404         {
7405             \dim_max:nn
7406             { \g_@@_blocks_dp_dim }
7407             {
7408                 \box_dp:c
7409                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7410             }
7411         }
7412     }
7413     \seq_gput_right:Ne \g_@@_blocks_seq
7414     {
7415         \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.

```

7416     {
7417         \exp_not:n { #3 } ,
7418         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7419         \bool_if:NT \g_@@_rotate_bool
7420         {
7421             \bool_if:NTF \g_@@_rotate_c_bool
7422             { m }
7423             {
7424                 \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7425                 { T }
7426             }
7427         }
7428     }
7429     {
7430         \box_use_drop:c
7431         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7432     }
7433 }
7434 \bool_set_false:N \g_@@_rotate_c_bool
7435 }

```

```

7436 \cs_new:Npn \@@_adjust_hpos_rotate:
7437 {
7438     \bool_if:NT \g_@@_rotate_bool
7439     {
7440         \str_set:Ne \l_@@_hpos_block_str
7441         {
7442             \bool_if:NTF \g_@@_rotate_c_bool
7443             { c }
7444             {

```

```

7445         \str_case:onF \l_@@_vpos_block_str
7446         { b l B l t r T r }
7447         {
7448             \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7449             { r }
7450             { l }
7451         }
7452     }
7453 }
7454 }
7455 }
7456 \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*.

```

7457 \cs_new_protected:Npn \@@_rotate_box_of_block:
7458 {
7459     \box_grotate:cn
7460     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7461     { 90 }
7462     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7463     {
7464         \vbox_gset_top:cn
7465         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7466         {
7467             \skip_vertical:n { 0.8 ex }
7468             \box_use:c
7469             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7470         }
7471     }
7472     \bool_if:NT \g_@@_rotate_c_bool
7473     {
7474         \hbox_gset:cn
7475         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7476         {
7477             \c_math_toggle_token
7478             \vcenter
7479             {
7480                 \box_use:c
7481                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7482             }
7483             \c_math_toggle_token
7484         }
7485     }
7486 }

```

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 now 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.

```

7487 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7488 {
7489     \seq_gput_right:Ne \g_@@_blocks_seq
7490     {
7491         \l_tmpa_tl
7492         { \exp_not:n { #3 } }
7493         {
7494             \bool_if:NTF \l_@@_tabular_bool
7495             {

```



```
7496 \group_begin:
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7497 \@@_reset_arraystretch:
7498 \exp_not:n
7499 {
7500   \dim_zero:N \extrarowheight
7501   #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).

```
7502   \bool_if:NT \c_@@_testphase_table_bool
7503     { \tag_stop:n { table } }
7504   \use:e
7505   {
7506     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7507     { @ { } \l_@@_hpos_block_str @ { } }
7508   }
7509   #5
7510   \end { tabular }
7511 }
7512 \group_end:
7513 }
```

When we are *not* in an environment `{NiceTabular}` (or similar).

```
7514 {
7515   \group_begin:
```

The following will be no-op when `respect-arraystretch` is in force.

```
7516 \@@_reset_arraystretch:
7517 \exp_not:n
7518 {
7519   \dim_zero:N \extrarowheight
7520   #4
7521   \c_math_toggle_token
7522   \use:e
7523   {
7524     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7525     { @ { } \l_@@_hpos_block_str @ { } }
7526   }
7527   #5
7528   \end { array }
7529   \c_math_toggle_token
7530 }
7531 \group_end:
7532 }
7533 }
7534 }
7535 }
7536 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
```

The following macro is for the case of a `\Block` which uses the key `p`.

```
7537 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7538 {
7539   \seq_gput_right:Ne \g_@@_blocks_seq
7540   {
7541     \l_tmpa_tl
7542     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```
7543 { { \exp_not:n { #4 #5 } } }
```

```

7544     }
7545   }
7546   \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }

```

The following macro is also for the case of a \Block which uses the key p.

```

7547   \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7548   {
7549     \seq_gput_right:Ne \g_@@_blocks_seq
7550     {
7551       \l_tmpa_tl
7552       { \exp_not:n { #3 } }
7553       { \exp_not:n { #4 #5 } }
7554     }
7555   }
7556   \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).

```

7557   \keys_define:nn { nicematrix / Block / SecondPass }
7558   {
7559     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7560     ampersand-in-blocks .default:n = true ,
7561     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence \l\_@@\_tikz\_seq will contain a sequence of comma-separated lists of keys.

```

7562     tikz .code:n =
7563       \IfPackageLoadedTF { tikz }
7564       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7565       { \@@_error:n { tikz~key~without~tikz } } ,
7566     tikz .value_required:n = true ,
7567     fill .code:n =
7568       \tl_set_rescan:Nnn
7569       \l_@@_fill_tl
7570       { \char_set_catcode_other:N ! }
7571       { #1 } ,
7572     fill .value_required:n = true ,
7573     opacity .tl_set:N = \l_@@_opacity_tl ,
7574     opacity .value_required:n = true ,
7575     draw .code:n =
7576       \tl_set_rescan:Nnn
7577       \l_@@_draw_tl
7578       { \char_set_catcode_other:N ! }
7579       { #1 } ,
7580     draw .default:n = default ,
7581     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7582     rounded-corners .default:n = 4 pt ,
7583     color .code:n =
7584       \@@_color:n { #1 }
7585       \tl_set_rescan:Nnn
7586       \l_@@_draw_tl
7587       { \char_set_catcode_other:N ! }
7588       { #1 } ,
7589     borders .clist_set:N = \l_@@_borders_clist ,
7590     borders .value_required:n = true ,
7591     hvlines .meta:n = { vlines , hlines } ,
7592     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7593     vlines .default:n = true ,
7594     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7595     hlines .default:n = true ,
7596     line-width .dim_set:N = \l_@@_line_width_dim ,
7597     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7598   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7599           \bool_set_true:N \l_@@_p_block_bool ,
7600   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7601   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7602   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7603   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7604           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7605   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7606           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7607   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7608           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7609   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7610   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7611   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7612   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7613   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7614   m .value_forbidden:n = true ,
7615   v-center .meta:n = m ,
7616   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7617   p .value_forbidden:n = true ,
7618   name .tl_set:N = \l_@@_block_name_str ,
7619   name .value_required:n = true ,
7620   name .initial:n = ,
7621   respect-arraystretch .code:n =
7622       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7623   respect-arraystretch .value_forbidden:n = true ,
7624   transparent .bool_set:N = \l_@@_transparent_bool ,
7625   transparent .default:n = true ,
7626   transparent .initial:n = false ,
7627   unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7628 }

```

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.

```

7629 \cs_new_protected:Npn \@@_draw_blocks:
7630 {
7631     \bool_if:NTF \c_@@_recent_array_bool
7632     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7633     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7634     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7635 }
7636 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7637 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7638     \int_zero:N \l_@@_last_row_int
7639     \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”).

```

7640     \int_compare:nNnTF { #3 } > { 98 }
7641     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7642     { \int_set:Nn \l_@@_last_row_int { #3 } }
7643     \int_compare:nNnTF { #4 } > { 98 }
7644     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7645     { \int_set:Nn \l_@@_last_col_int { #4 } }

```

```

7646 \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7647 {
7648   \bool_lazy_and:nnTF
7649     { \l_@@_preamble_bool }
7650     {
7651       \int_compare_p:n
7652         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7653     }
7654     {
7655       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7656       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7657       \@@_msg_redirect_name:nn { columns-not-used } { none }
7658     }
7659     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7660 }
7661 {
7662   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7663   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7664   {
7665     \@@_Block_v:nneenn
7666       { #1 }
7667       { #2 }
7668       { \int_use:N \l_@@_last_row_int }
7669       { \int_use:N \l_@@_last_col_int }
7670       { #5 }
7671       { #6 }
7672   }
7673 }
7674 }

```

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

```

7675 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7676 {

```

The group is for the keys.

```

7677   \group_begin:
7678   \int_compare:nNnT { #1 } = { #3 }
7679     { \str_set:Nn \l_@@_vpos_block_str { t } }
7680   \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

7681   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7682   \bool_lazy_and:nnT
7683     { \l_@@_vlines_block_bool }
7684     { ! \l_@@_ampersand_bool }
7685     {
7686       \tl_gput_right:Ne \g_nicematrix_code_after_tl
7687       {
7688         \@@_vlines_block:nnn
7689         { \exp_not:n { #5 } }
7690         { #1 - #2 }
7691         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7692       }
7693     }
7694   \bool_if:NT \l_@@_hlines_block_bool
7695     {
7696       \tl_gput_right:Ne \g_nicematrix_code_after_tl
7697       {
7698         \@@_hlines_block:nnn
7699         { \exp_not:n { #5 } }

```

```

7700         { #1 - #2 }
7701         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7702     }
7703 }
7704 \bool_if:NF \l_@@_transparent_bool
7705 {
7706     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7707     {

```

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).

```

7708         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7709         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7710     }
7711 }

7712 \tl_if_empty:NF \l_@@_draw_tl
7713 {
7714     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7715     { \@@_error:n { hlines~with~color } }
7716     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7717     {
7718         \@@_stroke_block:nnn

```

#5 are the options

```

7719         { \exp_not:n { #5 } }
7720         { #1 - #2 }
7721         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7722     }
7723     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7724     { { #1 } { #2 } { #3 } { #4 } }
7725 }

7726 \clist_if_empty:NF \l_@@_borders_clist
7727 {
7728     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7729     {
7730         \@@_stroke_borders_block:nnn
7731         { \exp_not:n { #5 } }
7732         { #1 - #2 }
7733         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7734     }
7735 }

7736 \tl_if_empty:NF \l_@@_fill_tl
7737 {
7738     \@@_add_opacity_to_fill:
7739     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7740     {
7741         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7742         { #1 - #2 }
7743         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7744         { \dim_use:N \l_@@_rounded_corners_dim }
7745     }
7746 }

7747 \seq_if_empty:NF \l_@@_tikz_seq
7748 {
7749     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7750     {
7751         \@@_block_tikz:nnnnn
7752         { \seq_use:Nn \l_@@_tikz_seq { , } }
7753         { #1 }
7754         { #2 }
7755         { \int_use:N \l_@@_last_row_int }
7756         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7757     }
7758 }

7759 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7760 {
7761     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7762     {
7763         \@@_actually_diagbox:nnnnnn
7764         { #1 }
7765         { #2 }
7766         { \int_use:N \l_@@_last_row_int }
7767         { \int_use:N \l_@@_last_col_int }
7768         { \exp_not:n { ##1 } }
7769         { \exp_not:n { ##2 } }
7770     }
7771 }

```

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.

```

7772 \pgfpicture
7773 \pgfrememberpicturepositiononpagetrue
7774 \pgf@relevantforpicturesizefalse
7775 \@@_qpoint:n { row - #1 }
7776 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7777 \@@_qpoint:n { col - #2 }
7778 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7779 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7780 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7781 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7782 \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.

```

7783 \@@_pgf_rect_node:nnnnn
7784 { \@@_env: - #1 - #2 - block }
7785 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7786 \str_if_empty:NF \l_@@_block_name_str
7787 {
7788     \pgfnodealias
7789     { \@@_env: - \l_@@_block_name_str }
7790     { \@@_env: - #1 - #2 - block }
7791     \str_if_empty:NF \l_@@_name_str

```

```

7792     {
7793         \pgfnodealias
7794         { \l_@@_name_str - \l_@@_block_name_str }
7795         { \@@_env: - #1 - #2 - block }
7796     }
7797 }

```

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.

```

7798     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7799     {
7800         \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.

```

7801         \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7802         {

```

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.

```

7803             \cs_if_exist:cT
7804             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7805             {
7806                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7807                 {
7808                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7809                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7810                 }
7811             }
7812         }

```

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.

```

7813         \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7814         {
7815             \@@_qpoint:n { col - #2 }
7816             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7817         }
7818     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7819     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7820     {
7821         \cs_if_exist:cT
7822         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7823         {
7824             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7825             {
7826                 \pgfpointanchor
7827                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7828                 { east }
7829                 \dim_set:Nn \l_@@_tmpd_dim
7830                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7831             }
7832         }
7833     }
7834     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7835     {
7836         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7837         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7838     }
7839     \@@_pgf_rect_node:nnnnn
7840     { \@@_env: - #1 - #2 - block - short }
7841     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim

```

7842 }

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.

```

7843 \bool_if:NT \l_@@_medium_nodes_bool
7844 {
7845   \@@pgf_rect_node:nnn
7846   { \@@_env: - #1 - #2 - block - medium }
7847   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7848   {
7849     \pgfpointanchor
7850     { \@@_env:
7851       - \int_use:N \l_@@_last_row_int
7852       - \int_use:N \l_@@_last_col_int - medium
7853     }
7854     { south-east }
7855   }
7856 }
7857 \endpgfpicture

```

```

7858 \bool_if:NTF \l_@@_ampersand_bool
7859 {
7860   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7861   \int_zero_new:N \l_@@_split_int
7862   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7863   \pgfpicture
7864   \pgfrememberpicturepositiononpagetrue
7865   \pgf@relevantforpicturesizefalse
7866
7867   \@@_qpoint:n { row - #1 }
7868   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7869   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7870   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7871   \@@_qpoint:n { col - #2 }
7872   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7873   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7874   \dim_set:Nn \l_tmpb_dim
7875     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7876   \bool_lazy_or:nnT
7877     { \l_@@_vlines_block_bool }
7878     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7879     {
7880       \int_step_inline:nn { \l_@@_split_int - 1 }
7881       {
7882         \pgfpathmoveto
7883         {
7884           \pgfpoint
7885             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7886             { \l_@@_tmpc_dim }
7887         }
7888         \pgfpathlineto
7889         {
7890           \pgfpoint
7891             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7892             { \l_@@_tmpd_dim }
7893         }
7894         \CT@arc@
7895         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7896         \pgfsetrectcap
7897         \pgfusepathqstroke
7898       }
7899     }
7900   \@@_qpoint:n { row - #1 - base }

```



```

7901 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7902 \int_step_inline:nn { \l_@@_split_int }
7903 {
7904   \group_begin:
7905   \dim_set:Nn \col@sep
7906   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7907   \pgftransformshift
7908   {
7909     \pgfpoint
7910     {
7911       \l_tmpa_dim + ##1 \l_tmpb_dim -
7912       \str_case:on \l_@@_hpos_block_str
7913       {
7914         l { \l_tmpb_dim + \col@sep }
7915         c { 0.5 \l_tmpb_dim }
7916         r { \col@sep }
7917       }
7918     }
7919     { \l_@@_tmpc_dim }
7920   }
7921   \pgfset { inner~sep = \c_zero_dim }
7922   \pgfnode
7923   { rectangle }
7924   {
7925     \str_case:on \l_@@_hpos_block_str
7926     {
7927       c { base }
7928       l { base~west }
7929       r { base~east }
7930     }
7931   }
7932   { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7933   \group_end:
7934 }
7935 \endpgfpicture
7936 }

```

Now the case where there is no ampersand & in the content of the block.

```

7937 {
7938   \bool_if:NTF \l_@@_p_block_bool
7939   {

```

When the final user has used the key p, we have to compute the width.

```

7940 \pgfpicture
7941 \pgfrememberpicturepositiononpagetrue
7942 \pgf@relevantforpicturesizefalse
7943 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7944 {
7945   \@@_qpoint:n { col - #2 }
7946   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7947   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7948 }
7949 {
7950   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7951   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7952   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7953 }
7954 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7955 \endpgfpicture
7956 \hbox_set:Nn \l_@@_cell_box
7957 {
7958   \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7959   { \g_tmpb_dim }
7960   \str_case:on \l_@@_hpos_block_str

```

```

7961         { c \centering r \raggedleft l \raggedright j { } }
7962         #6
7963         \end { minipage }
7964     }
7965 }
7966 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7967 \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.

```

7968 \pgfpicture
7969 \pgfrememberpicturepositiononpagetrue
7970 \pgf@relevantforpicturesizefalse
7971 \bool_lazy_any:nTF
7972 {
7973     { \str_if_empty_p:N \l_@@_vpos_block_str }
7974     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7975     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7976     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7977 }
7978 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7979     \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`.

```

7980     \bool_if:nT \g_@@_last_col_found_bool
7981     {
7982         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
7983         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7984     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7985     \tl_set:Ne \l_tmpa_tl
7986     {
7987         \str_case:on \l_@@_vpos_block_str
7988         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7989         { } {
7990             \str_case:on \l_@@_hpos_block_str
7991             {
7992                 c { center }
7993                 l { west }
7994                 r { east }
7995                 j { center }
7996             }
7997         }
7998     c {
7999         \str_case:on \l_@@_hpos_block_str
8000         {
8001             c { center }
8002             l { west }
8003             r { east }
8004             j { center }
8005         }
8006     }
8007 }
8008 T {
8009     \str_case:on \l_@@_hpos_block_str
8010     {
8011         c { north }

```

```

8012             l { north-west }
8013             r { north-east }
8014             j { north }
8015         }
8016
8017     }
8018     B {
8019         \str_case:on \l_@@_hpos_block_str
8020         {
8021             c { south }
8022             l { south-west }
8023             r { south-east }
8024             j { south }
8025         }
8026
8027     }
8028 }
8029
8030 \pgftransformshift
8031 {
8032     \pgfpointanchor
8033     {
8034         \@@_env: - #1 - #2 - block
8035         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8036     }
8037     { \l_tmpa_tl }
8038 }
8039 \pgfset { inner-sep = \c_zero_dim }
8040 \pgfnode
8041 { rectangle }
8042 { \l_tmpa_tl }
8043 { \box_use_drop:N \l_@@_cell_box } { } { }
8044 }

```

End of the case when  $\l_@@\_vpos\_block\_str$  is equal to c, T or B. Now, the other cases.

```

8045 {
8046     \pgfextracty \l_tmpa_dim
8047     {
8048         \@@_qpoint:n
8049         {
8050             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8051             - base
8052         }
8053     }
8054     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in  $\pgf@x$ ) the  $x$ -value of the center of the block.

```

8055 \pgfpointanchor
8056 {
8057     \@@_env: - #1 - #2 - block
8058     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8059 }
8060 {
8061     \str_case:on \l_@@_hpos_block_str
8062     {
8063         c { center }
8064         l { west }
8065         r { east }
8066         j { center }
8067     }
8068 }

```

We put the label of the block which has been composed in  $\l_@@\_cell\_box$ .

```

8069 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }

```

```

8070         \pgfset { inner~sep = \c_zero_dim }
8071         \pgfnode
8072         { rectangle }
8073         {
8074             \str_case:on \l_@@_hpos_block_str
8075             {
8076                 c { base }
8077                 l { base~west }
8078                 r { base~east }
8079                 j { base }
8080             }
8081         }
8082         { \box_use_drop:N \l_@@_cell_box } { } { }
8083     }
8084     \endpgfpicture
8085 }
8086 \group_end:
8087 }
8088 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8089 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8090 {
8091     \pgfpicture
8092     \pgfrememberpicturepositiononpagetrue
8093     \pgf@relevantforpicturesizefalse
8094     \pgfpathrectanglecorners
8095     { \pgfpoint { #2 } { #3 } }
8096     { \pgfpoint { #4 } { #5 } }
8097     \pgfsetfillcolor { #1 }
8098     \pgfusepath { fill }
8099     \endpgfpicture
8100 }

```

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).

```

8101 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8102 {
8103     \tl_if_empty:NF \l_@@_opacity_tl
8104     {
8105         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8106             {
8107                 \tl_set:Ne \l_@@_fill_tl
8108                 {
8109                     [ opacity = \l_@@_opacity_tl ,
8110                     \tl_tail:o \l_@@_fill_tl
8111                 }
8112             }
8113             {
8114                 \tl_set:Ne \l_@@_fill_tl
8115                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8116             }
8117         }
8118     }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8119 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3

```

```

8120 {
8121   \group_begin:
8122   \tl_clear:N \l_@@_draw_tl
8123   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8124   \keys_set_known:n { nicematrix / BlockStroke } { #1 }
8125   \pgfpicture
8126   \pgfrememberpicturepositiononpagetrue
8127   \pgf@relevantforpicturesizefalse
8128   \tl_if_empty:NF \l_@@_draw_tl
8129   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8130     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8131     { \CT@arc@ }
8132     { \@@_color:o \l_@@_draw_tl }
8133   }
8134   \pgfsetcornersarced
8135   {
8136     \pgfpoint
8137     { \l_@@_rounded_corners_dim }
8138     { \l_@@_rounded_corners_dim }
8139   }
8140   \@@_cut_on_hyphen:w #2 \q_stop
8141   \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8142   {
8143     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8144     {
8145       \@@_qpoint:n { row - \l_tmpa_tl }
8146       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8147       \@@_qpoint:n { col - \l_tmpb_tl }
8148       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8149       \@@_cut_on_hyphen:w #3 \q_stop
8150       \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8151       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8152       \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8153       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8154       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8155       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8156       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8157       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8158       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8159       \pgfpathrectanglecorners
8160       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8161       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8162       \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8163       { \pgfusepathqstroke }
8164       { \pgfusepath { stroke } }
8165     }
8166   }
8167   \endpgfpicture
8168   \group_end:
8169 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8170 \keys_define:nn { nicematrix / BlockStroke }
8171 {
8172   color .tl_set:N = \l_@@_draw_tl ,
8173   draw .code:n =
8174     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8175   draw .default:n = default ,
8176   line-width .dim_set:N = \l_@@_line_width_dim ,
8177   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8178   rounded-corners .default:n = 4 pt

```

8179 }

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax  $i-j$ ) and the third is the last cell of the block (with the same syntax).

```

8180 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8181 {
8182   \group_begin:
8183   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8184   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8185   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8186   \@@_cut_on_hyphen:w #2 \q_stop
8187   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8188   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8189   \@@_cut_on_hyphen:w #3 \q_stop
8190   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8191   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8192   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8193   {
8194     \use:e
8195     {
8196       \@@_vline:n
8197       {
8198         position = ##1 ,
8199         start = \l_@@_tmpc_tl ,
8200         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8201         total-width = \dim_use:N \l_@@_line_width_dim
8202       }
8203     }
8204   }
8205   \group_end:
8206 }

```

```

8207 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8208 {
8209   \group_begin:
8210   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8211   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8212   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8213   \@@_cut_on_hyphen:w #2 \q_stop
8214   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8215   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8216   \@@_cut_on_hyphen:w #3 \q_stop
8217   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8218   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8219   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8220   {
8221     \use:e
8222     {
8223       \@@_hline:n
8224       {
8225         position = ##1 ,
8226         start = \l_@@_tmpd_tl ,
8227         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8228         total-width = \dim_use:N \l_@@_line_width_dim
8229       }
8230     }
8231   }
8232   \group_end:
8233 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax  $i-j$ )

and the third is the last cell of the block (with the same syntax).

```

8234 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8235 {
8236   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8237   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8238   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8239     { \@@_error:n { borders~forbidden } }
8240     {
8241       \tl_clear_new:N \l_@@_borders_tikz_tl
8242       \keys_set:no
8243         { nicematrix / OnlyForTikzInBorders }
8244         \l_@@_borders_clist
8245         \@@_cut_on_hyphen:w #2 \q_stop
8246         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8247         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8248         \@@_cut_on_hyphen:w #3 \q_stop
8249         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8250         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8251         \@@_stroke_borders_block_i:
8252     }
8253 }

8254 \hook_gput_code:nnn { begindocument } { . }
8255 {
8256   \cs_new_protected:Npe \@@_stroke_borders_block_ii:
8257   {
8258     \c_@@_pgfortikzpicture_tl
8259     \@@_stroke_borders_block_ii:
8260     \c_@@_endpgfortikzpicture_tl
8261   }
8262 }

8263 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8264 {
8265   \pgfrememberpicturepositiononpagetrue
8266   \pgf@relevantforpicturesizefalse
8267   \CT@arc@
8268   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8269   \clist_if_in:NnT \l_@@_borders_clist { right }
8270     { \@@_stroke_vertical:n \l_tmpb_tl }
8271   \clist_if_in:NnT \l_@@_borders_clist { left }
8272     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8273   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8274     { \@@_stroke_horizontal:n \l_tmpa_tl }
8275   \clist_if_in:NnT \l_@@_borders_clist { top }
8276     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8277 }

8278 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8279 {
8280   tikz .code:n =
8281     \cs_if_exist:NTF \tikzpicture
8282       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8283       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8284   tikz .value_required:n = true ,
8285   top .code:n = ,
8286   bottom .code:n = ,
8287   left .code:n = ,
8288   right .code:n = ,
8289   unknown .code:n = \@@_error:n { bad~border }
8290 }

```

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).

```

8291 \cs_new_protected:Npn \@@_stroke_vertical:n #1

```

```

8292 {
8293   \@@_qpoint:n \l_@@_tmpc_tl
8294   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8295   \@@_qpoint:n \l_tmpa_tl
8296   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8297   \@@_qpoint:n { #1 }
8298   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8299   {
8300     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8301     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8302     \pgfusepathqstroke
8303   }
8304   {
8305     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8306     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8307   }
8308 }

```

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).

```

8309 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8310 {
8311   \@@_qpoint:n \l_@@_tmpd_tl
8312   \clist_if_in:NnTF \l_@@_borders_clist { left }
8313   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8314   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8315   \@@_qpoint:n \l_tmpb_tl
8316   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8317   \@@_qpoint:n { #1 }
8318   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8319   {
8320     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8321     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8322     \pgfusepathqstroke
8323   }
8324   {
8325     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8326     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8327   }
8328 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8329 \keys_define:nn { nicematrix / BlockBorders }
8330 {
8331   borders .clist_set:N = \l_@@_borders_clist ,
8332   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8333   rounded-corners .default:n = 4 pt ,
8334   line-width .dim_set:N = \l_@@_line_width_dim
8335 }

```

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.

```

8336 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8337 {
8338   \begin { tikzpicture }
8339   \@@_clip_with_rounded_corners:

```



We use `clist_map_inline:nn` because #5 is a list of lists.

```
8340 \clist_map_inline:nn { #1 }
8341 {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8342 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8343 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8344 (
8345 [
8346 xshift = \dim_use:N \l_@@_offset_dim ,
8347 yshift = - \dim_use:N \l_@@_offset_dim
8348 ]
8349 #2 -| #3
8350 )
8351 rectangle
8352 (
8353 [
8354 xshift = - \dim_use:N \l_@@_offset_dim ,
8355 yshift = \dim_use:N \l_@@_offset_dim
8356 ]
8357 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8358 ) ;
8359 }
8360 \end { tikzpicture }
8361 }
8362 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8363 \keys_define:nn { nicematrix / SpecialOffset }
8364 { offset .dim_set:N = \l_@@_offset_dim }
```

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.

```
8365 \cs_new_protected:Npn \@@_NullBlock:
8366 { \@@_collect_options:n { \@@_NullBlock_i: } }
8367 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8368 { }
```

## 27 How to draw the dotted lines transparently

```
8369 \cs_set_protected:Npn \@@_renew_matrix:
8370 {
8371 \RenewDocumentEnvironment { pmatrix } { } {
8372 { \pNiceMatrix }
8373 { \endpNiceMatrix }
8374 \RenewDocumentEnvironment { vmatrix } { } {
8375 { \vNiceMatrix }
8376 { \endvNiceMatrix }
8377 \RenewDocumentEnvironment { Vmatrix } { } {
8378 { \VNiceMatrix }
8379 { \endVNiceMatrix }
8380 \RenewDocumentEnvironment { bmatrix } { } {
8381 { \bNiceMatrix }
8382 { \endbNiceMatrix }
8383 \RenewDocumentEnvironment { Bmatrix } { } {
8384 { \BNiceMatrix }
8385 { \endBNiceMatrix }
8386 }
```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

8387 \keys_define:nn { nicematrix / Auto }
8388 {
8389   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8390   columns-type .value_required:n = true ,
8391   l .meta:n = { columns-type = l } ,
8392   r .meta:n = { columns-type = r } ,
8393   c .meta:n = { columns-type = c } ,
8394   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8395   delimiters / color .value_required:n = true ,
8396   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8397   delimiters / max-width .default:n = true ,
8398   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8399   delimiters .value_required:n = true ,
8400   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8401   rounded-corners .default:n = 4 pt
8402 }

8403 \NewDocumentCommand \AutoNiceMatrixWithDelims
8404 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8405 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8406 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8407 {

```

The group is for the protection of the keys.

```

8408   \group_begin:
8409   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8410   \use:e
8411   {
8412     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8413     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8414     [ \exp_not:o \l_tmpa_tl ]
8415   }
8416   \int_if_zero:nT { \l_@@_first_row_int }
8417   {
8418     \int_if_zero:nT { \l_@@_first_col_int } { & }
8419     \prg_replicate:nn { #4 - 1 } { & }
8420     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8421   }
8422   \prg_replicate:nn { #3 }
8423   {
8424     \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).

```

8425     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8426     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8427   }
8428   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8429   {
8430     \int_if_zero:nT { \l_@@_first_col_int } { & }
8431     \prg_replicate:nn { #4 - 1 } { & }
8432     \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8433   }
8434   \end { NiceArrayWithDelims }
8435   \group_end:
8436 }

8437 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8438 {

```

```

8439 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8440 {
8441   \bool_gset_true:N \g_@@_delims_bool
8442   \str_gset:N \g_@@_name_env_str { #1 AutoNiceMatrix }
8443   \AutoNiceMatrixWithDelims { #2 } { #3 }
8444 }
8445 }

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8446 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8447 {
8448   \group_begin:
8449   \bool_gset_false:N \g_@@_delims_bool
8450   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8451   \group_end:
8452 }

```

## 29 The redefinition of the command `\dotfill`

```

8453 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8454 \cs_new_protected:Npn \@@_dotfill:
8455 {

```

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}`).

```

8456   \@@_old_dotfill:
8457   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8458 }

```

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`.

```

8459 \cs_new_protected:Npn \@@_dotfill_i:
8460 {
8461   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8462   { \@@_old_dotfill: }
8463 }

```

## 30 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.

```

8464 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8465 {
8466   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8467   {
8468     \@@_actually_diagbox:nnnnnn
8469     { \int_use:N \c_iRow }
8470     { \int_use:N \c_jCol }
8471     { \int_use:N \c_iRow }
8472     { \int_use:N \c_jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8473         { \g_@@_row_style_tl \exp_not:n { #1 } }
8474         { \g_@@_row_style_tl \exp_not:n { #2 } }
8475     }

```

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`.

```

8476     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8477     {
8478         { \int_use:N \c@iRow }
8479         { \int_use:N \c@jCol }
8480         { \int_use:N \c@iRow }
8481         { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8482         { }
8483     }
8484 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_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.

```

8485 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8486 {
8487     \pgfpicture
8488     \pgf@relevantforpicturesizefalse
8489     \pgfrememberpicturepositiononpagetrue
8490     \@@_qpoint:n { row - #1 }
8491     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8492     \@@_qpoint:n { col - #2 }
8493     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8494     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8495     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8496     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8497     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8498     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8499     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8500 }

```

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.

```

8501     \CT@arc@
8502     \pgfsetroundcap
8503     \pgfusepathqstroke
8504 }
8505 \pgfset { inner~sep = 1 pt }
8506 \pgfscope
8507 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8508 \pgfnode { rectangle } { south~west }
8509 {
8510     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8511     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8512     \end { minipage }
8513 }
8514 { }
8515 { }
8516 \endpgfscope
8517 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8518 \pgfnode { rectangle } { north~east }
8519 {
8520     \begin { minipage } { 20 cm }
8521     \raggedleft

```

```

8522     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8523     \end { minipage }
8524   }
8525   { }
8526   { }
8527 \endpgfpicture
8528 }

```

## 31 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. 85.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8529 \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 `\`.

```

8530 \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`.

```

8531 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8532 {
8533   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8534   \@@_CodeAfter_iv:n
8535 }

```

We catch the argument of the command `\end` (in `#1`).

```

8536 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8537 {

```

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.

```

8538   \str_if_eq:eeTF \@currenvir { #1 }
8539   { \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`.

```

8540   {
8541     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8542     \@@_CodeAfter_ii:n
8543   }
8544 }

```

## 32 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.

```

8545 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8546 {
8547   \pgfpicture
8548   \pgfrememberpicturepositiononpagetrue
8549   \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.

```

8550   \@@_qpoint:n { row - 1 }
8551   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8552   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8553   \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).

```

8554   \bool_if:nTF { #3 }
8555   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8556   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8557   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8558   {
8559     \cs_if_exist:cT
8560     { \pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8561     {
8562       \pgfpointanchor
8563       { \@@_env: - ##1 - #2 }
8564       { \bool_if:nTF { #3 } { west } { east } }
8565       \dim_set:Nn \l_tmpa_dim
8566       {
8567         \bool_if:nTF { #3 }
8568         { \dim_min:nn }
8569         { \dim_max:nn }
8570         \l_tmpa_dim
8571         { \pgf@x }
8572       }
8573     }
8574   }

```

Now we can put the delimiter with a node of PGF.

```

8575   \pgfset { inner~sep = \c_zero_dim }
8576   \dim_zero:N \nulldelimiterspace
8577   \pgftransformshift
8578   {
8579     \pgfpoint
8580     { \l_tmpa_dim }
8581     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8582   }
8583   \pgfnode
8584   { rectangle }
8585   { \bool_if:nTF { #3 } { east } { west } }
8586   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8587     \nullfont
8588     \c_math_toggle_token
8589     \@@_color:o \l_@@_delimiters_color_tl
8590     \bool_if:nTF { #3 } { \left #1 } { \left . }
8591     \vcenter
8592     {
8593       \nullfont
8594       \hrule \@height

```

```

8595         \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8596         \@depth \c_zero_dim
8597         \@width \c_zero_dim
8598     }
8599     \bool_if:nTF { #3 } { \right . } { \right #1 }
8600     \c_math_toggle_token
8601 }
8602 { }
8603 { }
8604 \endpgfpicture
8605 }

```

### 33 The command `\SubMatrix`

```

8606 \keys_define:nn { nicematrix / sub-matrix }
8607 {
8608     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8609     extra-height .value_required:n = true ,
8610     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8611     left-xshift .value_required:n = true ,
8612     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8613     right-xshift .value_required:n = true ,
8614     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8615     xshift .value_required:n = true ,
8616     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8617     delimiters / color .value_required:n = true ,
8618     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8619     slim .default:n = true ,
8620     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8621     hlines .default:n = all ,
8622     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8623     vlines .default:n = all ,
8624     hvlines .meta:n = { hlines, vlines } ,
8625     hvlines .value_forbidden:n = true
8626 }
8627 \keys_define:nn { nicematrix }
8628 {
8629     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8630     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8631     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8632     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8633 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8634 \keys_define:nn { nicematrix / SubMatrix }
8635 {
8636     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8637     delimiters / color .value_required:n = true ,
8638     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8639     hlines .default:n = all ,
8640     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8641     vlines .default:n = all ,
8642     hvlines .meta:n = { hlines, vlines } ,
8643     hvlines .value_forbidden:n = true ,
8644     name .code:n =
8645         \tl_if_empty:nTF { #1 }
8646         { \@@_error:n { Invalid-name } }
8647         {
8648             \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }

```

```

8649         {
8650             \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8651             { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8652             {
8653                 \str_set:Nn \l_@@_submatrix_name_str { #1 }
8654                 \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8655             }
8656         }
8657         { \@@_error:n { Invalid~name } }
8658     } ,
8659     name .value_required:n = true ,
8660     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8661     rules .value_required:n = true ,
8662     code .tl_set:N = \l_@@_code_tl ,
8663     code .value_required:n = true ,
8664     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8665 }

8666 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8667 {
8668     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8669     {
8670         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8671         [
8672             delimiters / color = \l_@@_delimiters_color_tl ,
8673             hlines = \l_@@_submatrix_hlines_clist ,
8674             vlines = \l_@@_submatrix_vlines_clist ,
8675             extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8676             left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8677             right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8678             slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8679             #5
8680         ]
8681     }
8682     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8683     \ignorespaces
8684 }

8685 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8686 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8687 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8688 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8689 {
8690     \seq_gput_right:Ne \g_@@_submatrix_seq
8691     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8692     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8693     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8694     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8695     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8696 }
8697 }

```

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).

```

8698 \NewDocumentCommand \@@_compute_i_j:nn
8699 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8700 { \@@_compute_i_j:nnnn #1 #2 }

8701 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8702 {
8703     \def \l_@@_first_i_tl { #1 }

```



```

8704 \def \l_@@_first_j_tl { #2 }
8705 \def \l_@@_last_i_tl { #3 }
8706 \def \l_@@_last_j_tl { #4 }
8707 \tl_if_eq:NnT \l_@@_first_i_tl { last }
8708 { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8709 \tl_if_eq:NnT \l_@@_first_j_tl { last }
8710 { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8711 \tl_if_eq:NnT \l_@@_last_i_tl { last }
8712 { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8713 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8714 { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8715 }

```

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`.

```

8716 \hook_gput_code:nnn { begindocument } { . }
8717 {
8718   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8719   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8720     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8721 }
8722 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8723 {
8724   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8725 \@@_compute_i_j:nn { #2 } { #3 }
8726 \int_compare:NnNT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8727 { \def \arraystretch { 1 } }
8728 \bool_lazy_or:nnTF
8729 { \int_compare_p:Nn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8730 { \int_compare_p:Nn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8731 { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8732 {
8733   \str_clear_new:N \l_@@_submatrix_name_str
8734   \keys_set:nn { nicematrix / SubMatrix } { #5 }
8735   \pgfpicture
8736   \pgfrememberpicturepositiononpagetrue
8737   \pgf@relevantforpicturesizefalse
8738   \pgfset { inner~sep = \c_zero_dim }
8739   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8740   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currying.

```

8741 \bool_if:NTF \l_@@_submatrix_slim_bool
8742 { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8743 { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8744 {

```

```

8745 \cs_if_exist:cT
8746 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8747 {
8748 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8749 \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
8750 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8751 }
8752 \cs_if_exist:cT
8753 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8754 {
8755 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8756 \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
8757 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8758 }
8759 }
8760 \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
8761 { \@@_error:nn { Impossible~delimiter } { left } }
8762 {
8763 \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }
8764 { \@@_error:nn { Impossible~delimiter } { right } }
8765 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8766 }
8767 \endpgfpicture
8768 }
8769 \group_end:
8770 \ignorespaces
8771 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8772 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8773 {
8774 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8775 \dim_set:Nn \l_@@_y_initial_dim
8776 {
8777 \fp_to_dim:n
8778 {
8779 \pgf@y
8780 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8781 }
8782 }
8783 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8784 \dim_set:Nn \l_@@_y_final_dim
8785 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8786 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8787 {
8788 \cs_if_exist:cT
8789 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8790 {
8791 \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8792 \dim_set:Nn \l_@@_y_initial_dim
8793 { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8794 }
8795 \cs_if_exist:cT
8796 { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8797 {
8798 \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8799 \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8800 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8801 }
8802 }
8803 \dim_set:Nn \l_tmpa_dim
8804 {
8805 \l_@@_y_initial_dim - \l_@@_y_final_dim +

```

```

8806         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8807     }
8808     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8809     \group_begin:
8810     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8811     \@@_set_CTarc:o \l_@@_rules_color_tl
8812     \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`.

```

8813     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8814     {
8815         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8816         {
8817             \int_compare:nNnT
8818             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8819             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8820                 \@@_qpoint:n { col - ##1 }
8821                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8822                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8823                 \pgfusepathqstroke
8824             }
8825         }
8826     }

```

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.

```

8827     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8828     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8829     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8830     {
8831         \bool_lazy_and:nnTF
8832         { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8833         {
8834             \int_compare_p:nNn
8835             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8836         {
8837             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8838             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8839             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8840             \pgfusepathqstroke
8841         }
8842         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8843     }

```

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.

```

8844     \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8845     { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8846     { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8847     {
8848         \bool_lazy_and:nnTF
8849         { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8850         {
8851             \int_compare_p:nNn
8852             { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8853         {
8854             \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8855         \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
8856         \dim_set:Nn \l_tmpa_dim
8857         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8858         \str_case:nn { #1 }
8859         {
8860             ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8861             [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8862             \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8863             }
8864         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8865         \dim_set:Nn \l_tmpb_dim
8866         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8867         \str_case:nn { #2 }
8868         {
8869             ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8870             ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8871             \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8872         }
8873         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8874         \pgfusepathqstroke
8875         \group_end:
8876     }
8877     { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8878 }

```

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).

```

8879     \str_if_empty:NF \l_@@_submatrix_name_str
8880     {
8881         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8882         \l_@@_x_initial_dim \l_@@_y_initial_dim
8883         \l_@@_x_final_dim \l_@@_y_final_dim
8884     }
8885     \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`.

```

8886     \begin { pgfscope }
8887     \pgftransformshift
8888     {
8889         \pgfpoint
8890         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8891         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8892     }
8893     \str_if_empty:NTF \l_@@_submatrix_name_str
8894     { \@@_node_left:nn #1 { } }
8895     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8896     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8897     \pgftransformshift
8898     {
8899         \pgfpoint
8900         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8901         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8902     }
8903     \str_if_empty:NTF \l_@@_submatrix_name_str

```

```

8904 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8905 {
8906   \@@_node_right:nnnn #2
8907   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8908 }

```

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`.

```

8909 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8910 \flag_clear_new:N \l_@@_code_flag
8911 \l_@@_code_tl
8912 }

```

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`.

```

8913 \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.

```

8914 \cs_new:Npn \@@_pgfpointanchor:n #1
8915 { \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`).

```

8916 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8917 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
8918 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8919 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8920 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

8921 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

8922 { \@@_pgfpointanchor_ii:n { #1 } }
8923 }

```

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`).

```

8924 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8925 { \@@_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`.

```

8926 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

```

```

8927 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8928 {

```

The command `\str_if_empty:nTF` is “fully expandable”.

```

8929 \str_if_empty:nTF { #2 }

```

First the case where the argument does *not* contain an hyphen.

```

8930 { \@@_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`).

```

8931 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8932 }

```

The following function is for the case when the name contains an hyphen.

```

8933 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8934 {

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8935 \@@_env:
8936 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8937 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8938 }

```

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.

```

8939 \tl_const:Nn \c_@@_integers_alist_tl
8940 {
8941   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8942   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8943   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8944   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8945 }

```

```

8946 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8947 {

```

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`.

```

8948 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8949 {
8950   \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8951 \@@_env: -
8952 \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8953   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8954   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8955 }
8956 {
8957   \str_if_eq:eeTF { #1 } { last }
8958   {
8959     \flag_raise:N \l_@@_code_flag
8960     \@@_env: -
8961     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8962       { \int_eval:n { \l_@@_last_i_tl + 1 } }
8963       { \int_eval:n { \l_@@_last_j_tl + 1 } }
8964   }

```

```

8965         { #1 }
8966     }
8967 }

```

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`).

```

8968 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8969 {
8970     \pgfnode
8971     { rectangle }
8972     { east }
8973     {
8974         \nullfont
8975         \c_math_toggle_token
8976         \@@_color:o \l_@@_delimiters_color_tl
8977         \left #1
8978         \vcenter
8979         {
8980             \nullfont
8981             \hrule \@height \l_tmpa_dim
8982                 \@depth \c_zero_dim
8983                 \@width \c_zero_dim
8984         }
8985         \right .
8986         \c_math_toggle_token
8987     }
8988     { #2 }
8989     { }
8990 }

```

The command `\@@_node_right:nnn` 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.

```

8991 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8992 {
8993     \pgfnode
8994     { rectangle }
8995     { west }
8996     {
8997         \nullfont
8998         \c_math_toggle_token
8999         \colorlet { current-color } { . }
9000         \@@_color:o \l_@@_delimiters_color_tl
9001         \left .
9002         \vcenter
9003         {
9004             \nullfont
9005             \hrule \@height \l_tmpa_dim
9006                 \@depth \c_zero_dim
9007                 \@width \c_zero_dim
9008         }
9009         \right #1
9010         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9011         ~ { \color { current-color } \smash { #4 } }
9012         \c_math_toggle_token
9013     }
9014     { #2 }
9015     { }
9016 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9017 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9018 {
9019   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9020   \ignorespaces
9021 }

9022 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9023 {
9024   \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9025   \ignorespaces
9026 }

9027 \keys_define:nn { nicematrix / Brace }
9028 {
9029   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9030   left-shorten .default:n = true ,
9031   left-shorten .value_forbidden:n = true ,
9032   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9033   right-shorten .default:n = true ,
9034   right-shorten .value_forbidden:n = true ,
9035   shorten .meta:n = { left-shorten , right-shorten } ,
9036   shorten .value_forbidden:n = true ,
9037   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9038   yshift .value_required:n = true ,
9039   yshift .initial:n = \c_zero_dim ,
9040   color .tl_set:N = \l_tmpa_tl ,
9041   color .value_required:n = true ,
9042   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9043 }

```

**#1** is the first cell of the rectangle (with the syntax *i*-|*j*; **#2** is the last cell of the rectangle; **#3** is the label of the text; **#4** is the optional argument (a list of *key-value* pairs); **#5** is equal to `under` or `over`.

```

9044 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9045 {
9046   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9047 \@@_compute_i_j:nn { #1 } { #2 }
9048 \bool_lazy_or:nnTF
9049   { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9050   { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9051   {
9052     \str_if_eq:eeTF { #5 } { under }
9053     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9054     { \@@_error:nn { Construct-too-large } { \OverBrace } }
9055   }
9056   {
9057     \tl_clear:N \l_tmpa_tl
9058     \keys_set:nn { nicematrix / Brace } { #4 }
9059     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9060     \pgfpicture
9061     \pgfrememberpicturepositiononpagetrue
9062     \pgf@relevantforpicturesizefalse
9063     \bool_if:NT \l_@@_brace_left_shorten_bool
9064     {
9065       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9066       \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9067       {

```



```

9068         \cs_if_exist:cT
9069         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9070         {
9071             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9072
9073             \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9074             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9075         }
9076     }
9077 }
9078 \bool_lazy_or:nnT
9079 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9080 { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9081 {
9082     \@@_qpoint:n { col - \l_@@_first_j_tl }
9083     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9084 }
9085 \bool_if:NT \l_@@_brace_right_shorten_bool
9086 {
9087     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9088     \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9089     {
9090         \cs_if_exist:cT
9091         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9092         {
9093             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9094             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9095             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9096         }
9097     }
9098 }
9099 \bool_lazy_or:nnT
9100 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9101 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9102 {
9103     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9104     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9105 }
9106 \pgfset { inner~sep = \c_zero_dim }
9107 \str_if_eq:eeTF { #5 } { under }
9108 { \@@_underbrace_i:n { #3 } }
9109 { \@@_overbrace_i:n { #3 } }
9110 \endpgfpicture
9111 }
9112 \group_end:
9113 }

```

The argument is the text to put above the brace.

```

9114 \cs_new_protected:Npn \@@_overbrace_i:n #1
9115 {
9116     \@@_qpoint:n { row - \l_@@_first_i_tl }
9117     \pgftransformshift
9118     {
9119         \pgfpoint
9120         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9121         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9122     }
9123     \pgfnode
9124     { rectangle }
9125     { south }
9126     {
9127         \vtop
9128         {
9129             \group_begin:

```

```

9130     \everycr { }
9131     \halign
9132     {
9133         \hfil ## \hfil \crcr
9134         \bool_if:NTF \l_@@_tabular_bool
9135         { \begin { tabular } { c } #1 \end { tabular } }
9136         { $ \begin { array } { c } #1 \end { array } $ }
9137     \cr
9138     \c_math_toggle_token
9139     \overbrace
9140     {
9141         \hbox_to_wd:nn
9142         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9143         { }
9144     }
9145     \c_math_toggle_token
9146     \cr
9147     }
9148     \group_end:
9149 }
9150 }
9151 { }
9152 { }
9153 }

```

The argument is the text to put under the brace.

```

9154 \cs_new_protected:Npn \@@_underbrace_i:n #1
9155 {
9156     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9157     \pgftransformshift
9158     {
9159         \pgfpoint
9160         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9161         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9162     }
9163     \pgfnode
9164     { rectangle }
9165     { north }
9166     {
9167         \group_begin:
9168         \everycr { }
9169         \vbox
9170         {
9171             \halign
9172             {
9173                 \hfil ## \hfil \crcr
9174                 \c_math_toggle_token
9175                 \underbrace
9176                 {
9177                     \hbox_to_wd:nn
9178                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9179                     { }
9180                 }
9181                 \c_math_toggle_token
9182                 \cr
9183                 \bool_if:NTF \l_@@_tabular_bool
9184                 { \begin { tabular } { c } #1 \end { tabular } }
9185                 { $ \begin { array } { c } #1 \end { array } $ }
9186                 \cr
9187             }
9188         }
9189         \group_end:
9190     }
9191     { }

```

```

9192     { }
9193 }

```

## 35 The commands HBrace et VBrace

```

9194 \hook_gput_code:nnn { begindocument } { . }
9195 {
9196   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9197   {
9198     \tikzset
9199     {
9200       nicematrix / brace / .style =
9201       {
9202         decoration = { brace , raise = -0.15 em } ,
9203         decorate ,
9204       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9205       nicematrix / mirrored-brace / .style =
9206       {
9207         nicematrix / brace ,
9208         decoration = mirror ,
9209       }
9210     }
9211   }
9212 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9213 \keys_define:nn { nicematrix / Hbrace }
9214 {
9215   color .code:n = ,
9216   horizontal-label .code:n = ,
9217   horizontal-labels .code:n = ,
9218   shorten .code:n = ,
9219   shorten-start .code:n = ,
9220   shorten-end .code:n = ,
9221   unknown .code:n = \@@_error:n { Unknown~key~for~Hbrace }
9222 }

```

Here we need an “fully expandable” command.

```

9223 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9224 {
9225   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9226   { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9227   { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9228 }

```

The following command must *not* be protected.

```

9229 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9230 {
9231   \int_compare:nNnTF { \c@iRow } < { \c_one_int }
9232   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9233   \str_if_eq:nnTF { #2 } { * }
9234   {
9235     \NiceMatrixOptions { nullify-dots }

```

```

9236         \Ldots
9237         [
9238             line-style = nicematrix / brace ,
9239             #1 ,
9240             up =
9241             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9242         ]
9243     }
9244     {
9245         \Hdotsfor
9246         [
9247             line-style = nicematrix / brace ,
9248             #1 ,
9249             up =
9250             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9251         ]
9252         { #2 }
9253     }
9254 }
9255 {
9256     \str_if_eq:nnTF { #2 } { * }
9257     {
9258         \NiceMatrixOptions { nullify-dots }
9259         \Ldots
9260         [
9261             line-style = nicematrix / mirrored-brace ,
9262             #1 ,
9263             down =
9264             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9265         ]
9266     }
9267     {
9268         \Hdotsfor
9269         [
9270             line-style = nicematrix / mirrored-brace ,
9271             #1 ,
9272             down =
9273             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9274         ]
9275         { #2 }
9276     }
9277 }
9278 \keys_set:nn { nicematrix / Hbrace } { #1 }
9279 }

```

Here we need an “fully expandable” command.

```

9280 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9281 {
9282     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9283     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9284     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9285 }

```

The following command must *not* be protected.

```

9286 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9287 {
9288     \int_if_zero:nTF { \c@jCol }
9289     {
9290         \str_if_eq:nnTF { #2 } { * }
9291         {
9292             \NiceMatrixOptions { nullify-dots }
9293             \Vdots
9294             [
9295                 line-style = nicematrix / mirrored-brace ,

```

```

9296         #1 ,
9297         down =
9298             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9299     ]
9300 }
9301 {
9302     \Vdotsfor
9303     [
9304         line-style = nicematrix / mirrored-brace ,
9305         #1 ,
9306         down =
9307             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9308     ]
9309     { #2 }
9310 }
9311 }
9312 {
9313     \str_if_eq:nnTF { #2 } { * }
9314     {
9315         \NiceMatrixOptions { nullify-dots }
9316         \Vdots
9317         [
9318             line-style = nicematrix / brace ,
9319             #1 ,
9320             up =
9321                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9322         ]
9323     }
9324     {
9325         \Vdotsfor
9326         [
9327             line-style = nicematrix / brace ,
9328             #1 ,
9329             up =
9330                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9331         ]
9332         { #2 }
9333     }
9334 }
9335 \keys_set:nn { nicematrix / Hbrace } { #1 }
9336 }

```

## 36 The command TikzEveryCell

```

9337 \bool_new:N \l_@@_not_empty_bool
9338 \bool_new:N \l_@@_empty_bool
9339
9340 \keys_define:nn { nicematrix / TikzEveryCell }
9341 {
9342     not-empty .code:n =
9343         \bool_lazy_or:nnTF
9344         { \l_@@_in_code_after_bool }
9345         { \g_@@_create_cell_nodes_bool }
9346         { \bool_set_true:N \l_@@_not_empty_bool }
9347         { \@@_error:n { detection-of-empty-cells } } ,
9348     not-empty .value_forbidden:n = true ,
9349     empty .code:n =
9350         \bool_lazy_or:nnTF
9351         { \l_@@_in_code_after_bool }
9352         { \g_@@_create_cell_nodes_bool }
9353         { \bool_set_true:N \l_@@_empty_bool }
9354         { \@@_error:n { detection-of-empty-cells } } ,

```

```

9355     empty .value_forbidden:n = true ,
9356     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9357 }
9358
9359
9360 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9361 {
9362     \IfPackageLoadedTF { tikz }
9363     {
9364         \group_begin:
9365         \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.

```

9366         \tl_set:Nn \l_tmpa_tl { { #2 } }
9367         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9368         { \@@_for_a_block:nnnnn #1 }
9369         \@@_all_the_cells:
9370         \group_end:
9371     }
9372     { \@@_error:n { TikzEveryCell~without~tikz } }
9373 }
9374
9375 \tl_new:N \l_@@_i_tl
9376 \tl_new:N \l_@@_j_tl
9377
9378
9379 \cs_new_protected:Nn \@@_all_the_cells:
9380 {
9381     \int_step_inline:nn \c@iRow
9382     {
9383         \int_step_inline:nn \c@jCol
9384         {
9385             \cs_if_exist:cF { cell - ##1 - #####1 }
9386             {
9387                 \clist_if_in:Nef \l_@@_corners_cells_clist
9388                 { ##1 - #####1 }
9389                 {
9390                     \bool_set_false:N \l_tmpa_bool
9391                     \cs_if_exist:cTF
9392                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9393                     {
9394                         \bool_if:NF \l_@@_empty_bool
9395                         { \bool_set_true:N \l_tmpa_bool }
9396                     }
9397                     {
9398                         \bool_if:NF \l_@@_not_empty_bool
9399                         { \bool_set_true:N \l_tmpa_bool }
9400                     }
9401                     \bool_if:NT \l_tmpa_bool
9402                     {
9403                         \@@_block_tikz:nnnnn
9404                         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9405                     }
9406                 }
9407             }
9408         }
9409     }
9410 }
9411
9412 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9413 {
9414     \bool_if:NF \l_@@_empty_bool
9415     {

```

```

9416 \@@_block_tikz:nnnn
9417 \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9418 }
9419 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9420 }
9421
9422 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9423 {
9424 \int_step_inline:nnn { #1 } { #3 }
9425 {
9426 \int_step_inline:nnn { #2 } { #4 }
9427 { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9428 }
9429 }

```

## 37 The command \ShowCellNames

```

9430 \NewDocumentCommand \@@_ShowCellNames { }
9431 {
9432 \bool_if:NT \l_@@_in_code_after_bool
9433 {
9434 \pgfpicture
9435 \pgfrememberpicturepositiononpagetrue
9436 \pgf@relevantforpicturesizefalse
9437 \pgfpathrectanglecorners
9438 { \@@_qpoint:n { 1 } }
9439 {
9440 \@@_qpoint:n
9441 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9442 }
9443 \pgfsetfillopacity { 0.75 }
9444 \pgfsetfillcolor { white }
9445 \pgfusepathqfill
9446 \endpgfpicture
9447 }
9448 \dim_gzero_new:N \g_@@_tmpc_dim
9449 \dim_gzero_new:N \g_@@_tmpd_dim
9450 \dim_gzero_new:N \g_@@_tmpe_dim
9451 \int_step_inline:nn { \c@iRow }
9452 {
9453 \bool_if:NTF \l_@@_in_code_after_bool
9454 {
9455 \pgfpicture
9456 \pgfrememberpicturepositiononpagetrue
9457 \pgf@relevantforpicturesizefalse
9458 }
9459 { \begin { pgfpicture } }
9460 \@@_qpoint:n { row - ##1 }
9461 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9462 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9463 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9464 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9465 \bool_if:NTF \l_@@_in_code_after_bool
9466 { \endpgfpicture }
9467 { \end { pgfpicture } }
9468 \int_step_inline:nn { \c@jCol }
9469 {
9470 \hbox_set:Nn \l_tmpa_box
9471 {
9472 \normalfont \Large \sfamily \bfseries
9473 \bool_if:NTF \l_@@_in_code_after_bool
9474 { \color { red } }

```

```

9475         { \color { red ! 50 } }
9476     ##1 - ####1
9477 }
9478 \bool_if:NTF \l_@@_in_code_after_bool
9479 {
9480     \pgfpicture
9481     \pgfrememberpicturepositiononpagetrue
9482     \pgf@relevantforpicturesizefalse
9483 }
9484 { \begin { pgfpicture } }
9485 \@@_qpoint:n { col - ####1 }
9486 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9487 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9488 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9489 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9490 \bool_if:NTF \l_@@_in_code_after_bool
9491 { \endpgfpicture }
9492 { \end { pgfpicture } }
9493 \fp_set:Nn \l_tmpa_fp
9494 {
9495     \fp_min:nn
9496     {
9497         \fp_min:nn
9498         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9499         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9500     }
9501     { 1.0 }
9502 }
9503 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9504 \pgfpicture
9505 \pgfrememberpicturepositiononpagetrue
9506 \pgf@relevantforpicturesizefalse
9507 \pgftransformshift
9508 {
9509     \pgfpoint
9510     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9511     { \dim_use:N \g_tmpa_dim }
9512 }
9513 \pgfnode
9514 { rectangle }
9515 { center }
9516 { \box_use:N \l_tmpa_box }
9517 { }
9518 { }
9519 \endpgfpicture
9520 }
9521 }
9522 }

```

## 38 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.

```
9523 \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.



```

9524 \bool_new:N \g_@@_footnote_bool
9525 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9526 {
9527   You-have-used-the-key~' \l_keys_key_str '~when~loading~nicematrix~
9528   but~that~key~is~unknown. \\
9529   It~will~be~ignored. \\
9530   For~a~list~of~the~available~keys,~type~H~<return>.
9531 }
9532 {
9533   The~available~keys~are~(in~alphabetic~order):~
9534   footnote,~
9535   footnotehyper,~
9536   messages-for-Overleaf,~
9537   renew-dots~and~
9538   renew-matrix.
9539 }
9540 \keys_define:nn { nicematrix }
9541 {
9542   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9543   renew-dots .value_forbidden:n = true ,
9544   renew-matrix .code:n = \@@_renew_matrix: ,
9545   renew-matrix .value_forbidden:n = true ,
9546   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9547   footnote .bool_set:N = \g_@@_footnote_bool ,
9548   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9549   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9550 }
9551 \ProcessKeyOptions

9552 \@@_msg_new:nn { footnote-with-footnotehyper~package }
9553 {
9554   You~can't~use~the~option~'footnote'~because~the~package~
9555   footnotehyper~has~already~been~loaded.~
9556   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9557   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9558   of~the~package~footnotehyper.\\
9559   The~package~footnote~won't~be~loaded.
9560 }
9561 \@@_msg_new:nn { footnotehyper-with-footnote~package }
9562 {
9563   You~can't~use~the~option~'footnotehyper'~because~the~package~
9564   footnote~has~already~been~loaded.~
9565   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9566   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9567   of~the~package~footnote.\\
9568   The~package~footnotehyper~won't~be~loaded.
9569 }

9570 \bool_if:NT \g_@@_footnote_bool
9571 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9572 \IfClassLoadedTF { beamer }
9573 { \bool_set_false:N \g_@@_footnote_bool }
9574 {
9575   \IfPackageLoadedTF { footnotehyper }
9576   { \@@_error:n { footnote-with-footnotehyper~package } }
9577   { \usepackage { footnote } }
9578 }
9579 }

```

```

9580 \bool_if:NT \g_@@_footnotehyper_bool
9581 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9582 \IfClassLoadedTF { beamer }
9583 { \bool_set_false:N \g_@@_footnote_bool }
9584 {
9585   \IfPackageLoadedTF { footnote }
9586   { \@@_error:n { footnotehyper~with~footnote~package } }
9587   { \usepackage { footnotehyper } }
9588 }
9589 \bool_set_true:N \g_@@_footnote_bool
9590 }

```

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}`.

## 39 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.

```

9591 \bool_new:N \l_@@_underscore_loaded_bool
9592 \IfPackageLoadedT { underscore }
9593 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9594 \hook_gput_code:nnn { begindocument } { . }
9595 {
9596   \bool_if:NF \l_@@_underscore_loaded_bool
9597   {
9598     \IfPackageLoadedT { underscore }
9599     { \@@_error:n { underscore~after~nicematrix } }
9600   }
9601 }

```

## 40 Error messages of the package

```

9602 \str_const:Ne \c_@@_available_keys_str
9603 {
9604   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9605   { For~a~list~of~the~available~keys,~type~H~<return>. }
9606   { }
9607 }
9608 \seq_new:N \g_@@_types_of_matrix_seq
9609 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9610 {
9611   NiceMatrix ,
9612   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9613 }
9614 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9615 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_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`.

```

9616 \cs_new_protected:Npn \@@_error_too_much_cols:
9617 {
9618   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9619   { \@@_fatal:nn { too-much-cols-for-array } }
9620   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9621   { \@@_fatal:n { too-much-cols-for-matrix } }
9622   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9623   { \@@_fatal:n { too-much-cols-for-matrix } }
9624   \bool_if:NF \l_@@_last_col_without_value_bool
9625   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9626 }

```

The following command must *not* be protected since it's used in an error message.

```

9627 \cs_new:Npn \@@_message_hdotsfor:
9628 {
9629   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9630   { ~Maybe-your-use-of~ \token_to_str:N \Hdotsfor \ is~incorrect. }
9631 }
9632 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9633 {
9634   Incompatible-options.\\
9635   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9636   The-output-will-not-be-reliable.
9637 }
9638 \@@_msg_new:nn { key-color-inside }
9639 {
9640   Key-deprecated.\\
9641   The-key~'color-inside'~(and-its-alias~'colortbl-like')~is-now-point-less~
9642   and-have-been-deprecated.\\
9643   You-won't-have-similar-message-till-the-end-of-the-document.
9644 }
9645 \@@_msg_new:nn { invalid-weight }
9646 {
9647   Unknown-key.\\
9648   The-key~' \l_keys_key_str '~of~your~column~X~is-unknown-and-will-be-ignored.
9649 }
9650 \@@_msg_new:nn { last-col-not-used }
9651 {
9652   Column-not-used.\\
9653   The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
9654   in~your~\@@_full_name_env: .~
9655   However,~you-can~go~on.
9656 }
9657 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9658 {
9659   Too-much-columns.\\
9660   In~the~row~ \int_eval:n { \c@iRow },~
9661   you-try~to~use~more~columns~
9662   than~allowed~by~your~ \@@_full_name_env: .
9663   \@@_message_hdotsfor: \
9664   The-maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9665   (plus~the~exterior~columns).~This~error~is~fatal.
9666 }
9667 \@@_msg_new:nn { too-much-cols-for-matrix }
9668 {
9669   Too-much-columns.\\
9670   In~the~row~ \int_eval:n { \c@iRow },~
9671   you-try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9672   \@@_message_hdotsfor: \
9673   Recall~that~the~maximal~number~of~columns~for~a~matrix~

```

```

9674 (excepted-the-potential-exterior-columns)-is-fixed-by-the-
9675 LaTeX-counter-'MaxMatrixCols'.~
9676 Its-current-value-is~ \int_use:N \c@MaxMatrixCols \
9677 (use~ \token_to_str:N \setcounter \ to-change-that-value).~
9678 This-error-is-fatal.
9679 }

9680 \@@_msg_new:nn { too-much-cols-for-array }
9681 {
9682   Too-much-columns.\\
9683   In-the-row~ \int_eval:n { \c@iRow } ,~
9684   ~you-try-to-use-more-columns-than-allowed-by-your~
9685   \@@_full_name_env: . \@@_message_hdotsfor: \ The-maximal-number-of-columns-is~
9686   \int_use:N \g_@@_static_num_of_col_int \
9687   \bool_if:nT
9688     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9689     { ~(plus-the-exterior-ones) }
9690   since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9691   This-error-is-fatal.
9692 }

9693 \@@_msg_new:nn { columns-not-used }
9694 {
9695   Columns-not-used.\\
9696   The-preamble-of-your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
9697   It-announces~ \int_use:N \g_@@_static_num_of_col_int \
9698   columns-but-you-only-used~ \int_use:N \c@jCol .\\
9699   The-columns-you-did-not-used-won't-be-created.\\
9700   You-won't-have-similar-warning-till-the-end-of-the-document.
9701 }

9702 \@@_msg_new:nn { empty-preamble }
9703 {
9704   Empty-preamble.\\
9705   The-preamble-of-your~ \@@_full_name_env: \ is-empty.\\
9706   This-error-is-fatal.
9707 }

9708 \@@_msg_new:nn { in-first-col }
9709 {
9710   Erroneous-use.\\
9711   You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9712   That-command-will-be-ignored.
9713 }

9714 \@@_msg_new:nn { in-last-col }
9715 {
9716   Erroneous-use.\\
9717   You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9718   That-command-will-be-ignored.
9719 }

9720 \@@_msg_new:nn { in-first-row }
9721 {
9722   Erroneous-use.\\
9723   You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9724   That-command-will-be-ignored.
9725 }

9726 \@@_msg_new:nn { in-last-row }
9727 {
9728   Erroneous-use.\\
9729   You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9730   That-command-will-be-ignored.
9731 }

9732 \@@_msg_new:nn { TopRule-without-booktabs }
9733 {

```

```

9734     Erroneous~use.\\
9735     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9736     That~command~will~be~ignored.
9737 }

9738 \@@_msg_new:nn { TopRule~without~tikz }
9739 {
9740     Erroneous~use.\\
9741     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9742     That~command~will~be~ignored.
9743 }

9744 \@@_msg_new:nn { caption~outside~float }
9745 {
9746     Key~caption~forbidden.\\
9747     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9748     environment~(such~as~\{table\}).~This~key~will~be~ignored.
9749 }

9750 \@@_msg_new:nn { short~caption~without~caption }
9751 {
9752     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9753     However,~your~'short~caption'~will~be~used~as~'caption'.
9754 }

9755 \@@_msg_new:nn { double~closing~delimiter }
9756 {
9757     Double~delimiter.\\
9758     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9759     delimiter.~This~delimiter~will~be~ignored.
9760 }

9761 \@@_msg_new:nn { delimiter~after~opening }
9762 {
9763     Double~delimiter.\\
9764     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9765     delimiter.~That~delimiter~will~be~ignored.
9766 }

9767 \@@_msg_new:nn { bad~option~for~line~style }
9768 {
9769     Bad~line~style.\\
9770     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9771     is~'standard'.~That~key~will~be~ignored.
9772 }

9773 \@@_msg_new:nn { corners~with~no~cell~nodes }
9774 {
9775     Incompatible~keys.\\
9776     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9777     is~in~force.\\
9778     If~you~go~on,~that~key~will~be~ignored.
9779 }

9780 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9781 {
9782     Incompatible~keys.\\
9783     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9784     is~in~force.\\
9785     If~you~go~on,~those~extra~nodes~won't~be~created.
9786 }

9787 \@@_msg_new:nn { Identical~notes~in~caption }
9788 {
9789     Identical~tabular~notes.\\
9790     You~can't~put~several~notes~with~the~same~content~in~
9791     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9792     If~you~go~on,~the~output~will~probably~be~erroneous.
9793 }

```

```

9794 \@@_msg_new:nn { tabularnote~below~the~tabular }
9795 {
9796   \token_to_str:N \tabularnote \ forbidden\\
9797   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9798   of~your~tabular~because~the~caption~will~be~composed~below~
9799   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9800   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9801   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9802   no~similar~error~will~raised~in~this~document.
9803 }
9804 \@@_msg_new:nn { Unknown~key~for~rules }
9805 {
9806   Unknown~key.\\
9807   There~is~only~two~keys~available~here:~width~and~color.\\
9808   Your~key~' \l_keys_key_str '~will~be~ignored.
9809 }
9810 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9811 {
9812   Unknown~key.\\
9813   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9814   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9815   and~ \token_to_str:N \Vbrace \ are:~'color',~
9816   'horizontal~label(s)',~'shorten'~'shorten~end'~
9817   and~'shorten~start'.
9818 }
9819 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9820 {
9821   Unknown~key.\\
9822   There~is~only~two~keys~available~here:~
9823   'empty'~and~'not~empty'.\\
9824   Your~key~' \l_keys_key_str '~will~be~ignored.
9825 }
9826 \@@_msg_new:nn { Unknown~key~for~rotate }
9827 {
9828   Unknown~key.\\
9829   The~only~key~available~here~is~'c'.\\
9830   Your~key~' \l_keys_key_str '~will~be~ignored.
9831 }
9832 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9833 {
9834   Unknown~key.\\
9835   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9836   It~you~go~on,~you~will~probably~have~other~errors. \\
9837   \c_@@_available_keys_str
9838 }
9839 {
9840   The~available~keys~are~(in~alphabetic~order):~
9841   ccommand,~
9842   color,~
9843   command,~
9844   dotted,~
9845   letter,~
9846   multiplicity,~
9847   sep~color,~
9848   tikz,~and~total~width.
9849 }
9850 \@@_msg_new:nnn { Unknown~key~for~xdots }
9851 {
9852   Unknown~key.\\
9853   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9854   \c_@@_available_keys_str

```

```

9855 }
9856 {
9857   The-available-keys-are-(in-alphabetic-order):~
9858   'color',~
9859   'horizontal(s)-labels',~
9860   'inter',~
9861   'line-style',~
9862   'radius',~
9863   'shorten',~
9864   'shorten-end'~and~'shorten-start'.
9865 }
9866 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9867 {
9868   Unknown-key.\\
9869   As-for-now,~there-is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9870   (and-you-try-to-use~' \l_keys_key_str ')\
9871   That-key-will-be-ignored.
9872 }
9873 \@@_msg_new:nn { label-without-caption }
9874 {
9875   You-can't-use-the-key~'label'~in-your~\{NiceTabular\}~because~
9876   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9877 }
9878 \@@_msg_new:nn { W-warning }
9879 {
9880   Line~ \msg_line_number: .~The-cell-is-too-wide-for-your-column~'W'~
9881   (row~ \int_use:N \c@iRow ).
9882 }
9883 \@@_msg_new:nn { Construct-too-large }
9884 {
9885   Construct-too-large.\\
9886   Your-command~ \token_to_str:N #1
9887   can't-be-drawn-because-your-matrix-is-too-small.\\
9888   That-command-will-be-ignored.
9889 }
9890 \@@_msg_new:nn { underscore-after-nicematrix }
9891 {
9892   Problem-with-'underscore'.\\
9893   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9894   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9895   ' \token_to_str:N \Cdots \token_to_str:N _
9896   \{ n \token_to_str:N \text \{ ~times \} \}' .
9897 }
9898 \@@_msg_new:nn { ampersand-in-light-syntax }
9899 {
9900   Ampersand-forbidden.\\
9901   You-can't-use-an-ampersand~( \token_to_str:N & )~to-separate-columns-because~
9902   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9903 }
9904 \@@_msg_new:nn { double-backslash-in-light-syntax }
9905 {
9906   Double-backslash-forbidden.\\
9907   You-can't-use~ \token_to_str:N \\
9908   ~to-separate-rows-because-the-key~'light-syntax'~
9909   is-in-force.~You-must-use-the-character~' \l_@@_end_of_row_tl '~
9910   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9911 }
9912 \@@_msg_new:nn { hlines-with-color }
9913 {
9914   Incompatible-keys.\\

```

```

9915     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for-a~
9916     \token_to_str:N \Block \ when-the-key-'color'~or~'draw'~is-used.\\
9917     However,~you-can-put~several~commands~ \token_to_str:N \Block.\\
9918     Your-key-will-be-discarded.
9919 }
9920 \@@_msg_new:nn { bad-value-for-baseline }
9921 {
9922     Bad-value-for-baseline.\\
9923     The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
9924     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9925     \int_use:N \g_@@_row_total_int \ or-equal-to-'t',~'c'~or~'b'~or-of~
9926     the-form-'line-i'.\\
9927     A~value-of~1~will-be-used.
9928 }
9929 \@@_msg_new:nn { detection-of-empty-cells }
9930 {
9931     Problem-with-'not-empty'\\
9932     For-technical-reasons,~you-must-activate~
9933     'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
9934     in-order-to-use-the-key~' \l_keys_key_str '.\\
9935     That-key-will-be-ignored.
9936 }
9937 \@@_msg_new:nn { siunitx-not-loaded }
9938 {
9939     siunitx-not-loaded\\
9940     You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\\
9941     That-error-is-fatal.
9942 }
9943 \@@_msg_new:nn { Invalid-name }
9944 {
9945     Invalid-name.\\
9946     You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
9947     \SubMatrix \ of-your~ \@@_full_name_env: .\\
9948     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9949     This-key-will-be-ignored.
9950 }
9951 \@@_msg_new:nn { Hbrace-not-allowed }
9952 {
9953     Command-not-allowed.\\
9954     You-can't-use-the-command~ \token_to_str:N #1
9955     because-you-have-not-loaded~
9956     \IfPackageLoadedTF { tikz }
9957     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
9958     { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9959     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9960     That-command-will-be-ignored.
9961 }
9962 \@@_msg_new:nn { Vbrace-not-allowed }
9963 {
9964     Command-not-allowed.\\
9965     You-can't-use-the-command~ \token_to_str:N \Vbrace \
9966     because-you-have-not-loaded-TikZ~
9967     and-the-TikZ-library~'decorations.pathreplacing'\\.\\
9968     Use: ~\token_to_str:N \usepackage \{tikz\}~
9969     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
9970     That-command-will-be-ignored.
9971 }
9972 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9973 {
9974     Wrong-line.\\
9975     You-try-to-draw-a-#1~line-of-number~'#2'~in-a~

```



```

9976     \token_to_str:N \SubMatrix \ of-your~ \@@_full_name_env: \ but-that~
9977     number-is-not-valid.~It-will-be-ignored.
9978 }
9979 \@@_msg_new:nn { Impossible-delimiter }
9980 {
9981     Impossible-delimiter.\\
9982     It's-impossible-to-draw-the-#1-delimiter-of-your~
9983     \token_to_str:N \SubMatrix \ because-all-the-cells-are-empty~
9984     in-that-column.
9985     \bool_if:NT \l_@@_submatrix_slim_bool
9986     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
9987     This~ \token_to_str:N \SubMatrix \ will-be-ignored.
9988 }
9989 \@@_msg_new:nnn { width-without-X-columns }
9990 {
9991     You-have-used-the-key-'width'~but-you-have-put-no-'X'~column-in~
9992     the-preamble~(' \g_@@_user_preamble_tl ')~of-your~ \@@_full_name_env: .\\
9993     That-key-will-be-ignored.
9994 }
9995 {
9996     This-message-is-the-message-'width-without-X-columns'~
9997     of-the-module-'nicematrix'.~
9998     The-experimented-users-can-disable-that-message-with~
9999     \token_to_str:N \msg_redirect_name:nnn .\\
10000 }
10001
10002 \@@_msg_new:nn { key-multiplicity-with-dotted }
10003 {
10004     Incompatible-keys. \\
10005     You-have-used-the-key-'multiplicity'~with-the-key-'dotted'~
10006     in-a-'custom-line'.~They-are-incompatible. \\
10007     The-key-'multiplicity'~will-be-discarded.
10008 }
10009 \@@_msg_new:nn { empty-environment }
10010 {
10011     Empty-environment.\\
10012     Your~ \@@_full_name_env: \ is-empty.~This-error-is-fatal.
10013 }
10014 \@@_msg_new:nn { No-letter-and-no-command }
10015 {
10016     Erroneous-use.\\
10017     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
10018     key-'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys-'command'~or~
10019     ~'ccommand'~(to-draw-horizontal-rules).\\
10020     However,~you-can-go-on.
10021 }
10022 \@@_msg_new:nn { Forbidden-letter }
10023 {
10024     Forbidden-letter.\\
10025     You-can't-use-the-letter~'#1'~for-a-customized-line.~
10026     It-will-be-ignored.\\
10027     The-forbidden-letters~are:~\c_@@_forbidden_letters_str
10028 }
10029 \@@_msg_new:nn { Several-letters }
10030 {
10031     Wrong-name.\\
10032     You-must-use-only-one-letter-as-value-for-the-key-'letter'~(and-you~
10033     have-used~' \l_@@_letter_str ').\\
10034     It-will-be-ignored.
10035 }

```

```

10036 \@@_msg_new:nn { Delimiter-with-small }
10037 {
10038   Delimiter~forbidden.\\
10039   You~can't~put~a~delimiter~in~the~preamble~of~your~
10040   \@@_full_name_env: \
10041   because~the~key~'small'~is~in~force.\\
10042   This~error~is~fatal.
10043 }
10044 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
10045 {
10046   Unknown~cell.\\
10047   Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10048   the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10049   can't~be~executed~because~a~cell~doesn't~exist.\\
10050   This~command~ \token_to_str:N \line \ will~be~ignored.
10051 }
10052 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
10053 {
10054   Duplicate~name.\\
10055   The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10056   in~this~ \@@_full_name_env: .\\
10057   This~key~will~be~ignored.\\
10058   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10059   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10060 }
10061 {
10062   The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10063   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10064 }
10065 \@@_msg_new:nn { r-or-l-with-preamble }
10066 {
10067   Erroneous~use.\\
10068   You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10069   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10070   your~ \@@_full_name_env: .\\
10071   This~key~will~be~ignored.
10072 }
10073 \@@_msg_new:nn { Hdotsfor-in-col-0 }
10074 {
10075   Erroneous~use.\\
10076   You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10077   the~array.~This~error~is~fatal.
10078 }
10079 \@@_msg_new:nn { bad-corner }
10080 {
10081   Bad~corner.\\
10082   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10083   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10084   This~specification~of~corner~will~be~ignored.
10085 }
10086 \@@_msg_new:nn { bad-border }
10087 {
10088   Bad~border.\\
10089   \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10090   (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10091   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10092   also~use~the~key~'tikz'
10093   \IfPackageLoadedF { tikz }
10094   { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10095   This~specification~of~border~will~be~ignored.
10096 }

```

```

10097 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10098 {
10099   TikZ~not~loaded.\\
10100   You~can't~use~ \token_to_str:N \TikzEveryCell \
10101   because~you~have~not~loaded~tikz.~
10102   This~command~will~be~ignored.
10103 }
10104 \@@_msg_new:nn { tikz~key~without~tikz }
10105 {
10106   TikZ~not~loaded.\\
10107   You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10108   \Block '~because~you~have~not~loaded~tikz.~
10109   This~key~will~be~ignored.
10110 }
10111 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10112 {
10113   Erroneous~use.\\
10114   In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10115   'last~col'~without~value.\\
10116   However,~you~can~go~on~for~this~time~
10117   (the~value~' \l_keys_value_tl '~will~be~ignored).
10118 }
10119 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10120 {
10121   Erroneous~use. \\
10122   In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10123   'last~col'~without~value. \\
10124   However,~you~can~go~on~for~this~time~
10125   (the~value~' \l_keys_value_tl '~will~be~ignored).
10126 }
10127 \@@_msg_new:nn { Block~too~large~1 }
10128 {
10129   Block~too~large. \\
10130   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
10131   too~small~for~that~block. \\
10132   This~block~and~maybe~others~will~be~ignored.
10133 }
10134 \@@_msg_new:nn { Block~too~large~2 }
10135 {
10136   Block~too~large. \\
10137   The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10138   \g_@@_static_num_of_col_int \
10139   columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10140   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
10141   (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10142   This~block~and~maybe~others~will~be~ignored.
10143 }
10144 \@@_msg_new:nn { unknown~column~type }
10145 {
10146   Bad~column~type. \\
10147   The~column~type~'#1'~in~your~ \@@_full_name_env: \
10148   is~unknown. \\
10149   This~error~is~fatal.
10150 }
10151 \@@_msg_new:nn { unknown~column~type~S }
10152 {
10153   Bad~column~type. \\
10154   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10155   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10156   load~that~package. \\
10157   This~error~is~fatal.

```

```

10158 }
10159 \@@_msg_new:nn { tabularnote~forbidden }
10160 {
10161   Forbidden~command. \\
10162   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10163   ~here.~This~command~is~available~only~in~
10164   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10165   the~argument~of~a~command~\token_to_str:N \caption \ included~
10166   in~an~environment~\{table\}. \\
10167   This~command~will~be~ignored.
10168 }
10169 \@@_msg_new:nn { borders~forbidden }
10170 {
10171   Forbidden~key.\\
10172   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10173   because~the~option~'rounded~corners'~
10174   is~in~force~with~a~non~zero~value.\\
10175   This~key~will~be~ignored.
10176 }
10177 \@@_msg_new:nn { bottomrule~without~booktabs }
10178 {
10179   booktabs~not~loaded.\\
10180   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10181   loaded~'booktabs'.\\
10182   This~key~will~be~ignored.
10183 }
10184 \@@_msg_new:nn { enumitem~not~loaded }
10185 {
10186   enumitem~not~loaded. \\
10187   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10188   ~because~you~haven't~loaded~'enumitem'. \\
10189   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10190   ignored~in~the~document.
10191 }
10192 \@@_msg_new:nn { tikz~without~tikz }
10193 {
10194   Tikz~not~loaded. \\
10195   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10196   loaded.~If~you~go~on,~that~key~will~be~ignored.
10197 }
10198 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10199 {
10200   Tikz~not~loaded. \\
10201   You~have~used~the~key~'tikz'~in~the~definition~of~a~
10202   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10203   You~can~go~on~but~you~will~have~another~error~if~you~actually~
10204   use~that~custom~line.
10205 }
10206 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10207 {
10208   Tikz~not~loaded. \\
10209   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10210   command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10211   That~key~will~be~ignored.
10212 }
10213 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10214 {
10215   Erroneous~use.\\
10216   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10217   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~

```

```

10218     The~key~ 'color'~will~be~discarded.
10219 }
10220 \@@_msg_new:nn { Wrong~last~row }
10221 {
10222     Wrong~number.\\
10223     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10224     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10225     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10226     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10227     without~value~(more~compilations~might~be~necessary).
10228 }
10229 \@@_msg_new:nn { Yet~in~env }
10230 {
10231     Nested~environments.\\
10232     Environments~of~nicematrix~can't~be~nested.\\
10233     This~error~is~fatal.
10234 }
10235 \@@_msg_new:nn { Outside~math~mode }
10236 {
10237     Outside~math~mode.\\
10238     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10239     (and~not~in~ \token_to_str:N \vcenter ).\\
10240     This~error~is~fatal.
10241 }
10242 \@@_msg_new:nn { One~letter~allowed }
10243 {
10244     Bad~name.\\
10245     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10246     you~have~used~' \l_keys_value_tl '.\\
10247     It~will~be~ignored.
10248 }
10249 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10250 {
10251     Environment~\{TabularNote\}~forbidden.\\
10252     You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10253     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10254     This~environment~\{TabularNote\}~will~be~ignored.
10255 }
10256 \@@_msg_new:nn { varwidth~not~loaded }
10257 {
10258     varwidth~not~loaded.\\
10259     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10260     loaded.\\
10261     Your~column~will~behave~like~'p'.
10262 }
10263 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10264 {
10265     Unknown~key.\\
10266     Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
10267     \c_@@_available_keys_str
10268 }
10269 {
10270     The~available~keys~are~(in~alphabetic~order):~
10271     color,~
10272     dotted,~
10273     multiplicity,~
10274     sep~color,~
10275     tikz,~and~total~width.
10276 }
10277

```

```

10278 \@@_msg_new:nnn { Unknown~key~for~Block }
10279 {
10280   Unknown~key. \\
10281   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10282   \token_to_str:N \Block . \\
10283   It~will~be~ignored. \\
10284   \c_@@_available_keys_str
10285 }
10286 {
10287   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10288   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10289   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10290   and~vlines.
10291 }
10292 \@@_msg_new:nnn { Unknown~key~for~Brace }
10293 {
10294   Unknown~key. \\
10295   The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
10296   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10297   It~will~be~ignored. \\
10298   \c_@@_available_keys_str
10299 }
10300 {
10301   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10302   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10303   right~shorten)~and~yshift.
10304 }
10305 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10306 {
10307   Unknown~key. \\
10308   The~key~' \l_keys_key_str '~is~unknown. \\
10309   It~will~be~ignored. \\
10310   \c_@@_available_keys_str
10311 }
10312 {
10313   The~available~keys~are~(in~alphabetic~order):~
10314   delimiters/color,~
10315   rules~(with~the~subkeys~'color'~and~'width'),~
10316   sub~matrix~(several~subkeys)~
10317   and~xdots~(several~subkeys).~
10318   The~latter~is~for~the~command~ \token_to_str:N \line .
10319 }
10320 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10321 {
10322   Unknown~key. \\
10323   The~key~' \l_keys_key_str '~is~unknown. \\
10324   It~will~be~ignored. \\
10325   \c_@@_available_keys_str
10326 }
10327 {
10328   The~available~keys~are~(in~alphabetic~order):~
10329   create~cell~nodes,~
10330   delimiters/color~and~
10331   sub~matrix~(several~subkeys).
10332 }
10333 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10334 {
10335   Unknown~key. \\
10336   The~key~' \l_keys_key_str '~is~unknown. \\
10337   That~key~will~be~ignored. \\
10338   \c_@@_available_keys_str
10339 }

```

```

10340 {
10341     The-available-keys-are~(in~alphabetic-order):~
10342     'delimiters/color',~
10343     'extra-height',~
10344     'hlines',~
10345     'hvlines',~
10346     'left-xshift',~
10347     'name',~
10348     'right-xshift',~
10349     'rules'~(with~the~subkeys~'color'~and~'width'),~
10350     'slim',~
10351     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10352     and~'right-xshift').\\
10353 }
10354 \@@_msg_new:nnn { Unknown~key~for~notes }
10355 {
10356     Unknown~key.\\
10357     The~key~' \l_keys_key_str '~is~unknown.\\
10358     That~key~will~be~ignored. \\
10359     \c_@@_available_keys_str
10360 }
10361 {
10362     The-available-keys-are~(in~alphabetic-order):~
10363     bottomrule,~
10364     code-after,~
10365     code-before,~
10366     detect-duplicates,~
10367     enumitem-keys,~
10368     enumitem-keys-para,~
10369     para,~
10370     label-in-list,~
10371     label-in-tabular~and~
10372     style.
10373 }
10374 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10375 {
10376     Unknown~key.\\
10377     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10378     \token_to_str:N \RowStyle . \\
10379     That~key~will~be~ignored. \\
10380     \c_@@_available_keys_str
10381 }
10382 {
10383     The-available-keys-are~(in~alphabetic-order):~
10384     bold,~
10385     cell-space-top-limit,~
10386     cell-space-bottom-limit,~
10387     cell-space-limits,~
10388     color,~
10389     fill~(alias:~rowcolor),~
10390     nb-rows,~
10391     opacity~and~
10392     rounded-corners.
10393 }
10394 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10395 {
10396     Unknown~key.\\
10397     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10398     \token_to_str:N \NiceMatrixOptions . \\
10399     That~key~will~be~ignored. \\
10400     \c_@@_available_keys_str
10401 }
10402 {

```

```

10403 The~available~keys~are~(in~alphabetic~order):~
10404 &-in-blocks,~
10405 allow-duplicate-names,~
10406 ampersand-in-blocks,~
10407 caption-above,~
10408 cell-space-bottom-limit,~
10409 cell-space-limits,~
10410 cell-space-top-limit,~
10411 code-for-first-col,~
10412 code-for-first-row,~
10413 code-for-last-col,~
10414 code-for-last-row,~
10415 corners,~
10416 custom-key,~
10417 create-extra-nodes,~
10418 create-medium-nodes,~
10419 create-large-nodes,~
10420 custom-line,~
10421 delimiters~(several~subkeys),~
10422 end-of-row,~
10423 first-col,~
10424 first-row,~
10425 hlines,~
10426 hvlines,~
10427 hvlines-except-borders,~
10428 last-col,~
10429 last-row,~
10430 left-margin,~
10431 light-syntax,~
10432 light-syntax-expanded,~
10433 matrix/columns-type,~
10434 no-cell-nodes,~
10435 notes~(several~subkeys),~
10436 nullify-dots,~
10437 pgf-node-code,~
10438 renew-dots,~
10439 renew-matrix,~
10440 respect-arraystretch,~
10441 rounded-corners,~
10442 right-margin,~
10443 rules~(with~the~subkeys~'color'~and~'width'),~
10444 small,~
10445 sub-matrix~(several~subkeys),~
10446 vlins,~
10447 xdots~(several~subkeys).
10448 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10449 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10450 {
10451   Unknown~key.\\
10452   The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10453   \{NiceArray\}. \\
10454   That~key~will~be~ignored. \\
10455   \c_@@_available_keys_str
10456 }
10457 {
10458   The~available~keys~are~(in~alphabetic~order):~
10459   &-in-blocks,~
10460   ampersand-in-blocks,~
10461   b,~
10462   baseline,~
10463   c,~

```



```

10464     cell-space-bottom-limit,~
10465     cell-space-limits,~
10466     cell-space-top-limit,~
10467     code-after,~
10468     code-for-first-col,~
10469     code-for-first-row,~
10470     code-for-last-col,~
10471     code-for-last-row,~
10472     columns-width,~
10473     corners,~
10474     create-extra-nodes,~
10475     create-medium-nodes,~
10476     create-large-nodes,~
10477     extra-left-margin,~
10478     extra-right-margin,~
10479     first-col,~
10480     first-row,~
10481     hlines,~
10482     hvlines,~
10483     hvlines-except-borders,~
10484     last-col,~
10485     last-row,~
10486     left-margin,~
10487     light-syntax,~
10488     light-syntax-expanded,~
10489     name,~
10490     no-cell-nodes,~
10491     nullify-dots,~
10492     pgf-node-code,~
10493     renew-dots,~
10494     respect-arraystretch,~
10495     right-margin,~
10496     rounded-corners,~
10497     rules~(with~the~subkeys~'color'~and~'width'),~
10498     small,~
10499     t,~
10500     vlines,~
10501     xdots/color,~
10502     xdots/shorten-start,~
10503     xdots/shorten-end,~
10504     xdots/shorten-and~
10505     xdots/line-style.
10506 }

```

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`).

```

10507 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10508 {
10509     Unknown~key.\\
10510     The~key~' \l_keys_key_str '~is~unknown~for~the~
10511     \@@_full_name_env: . \\
10512     That~key~will~be~ignored. \\
10513     \c_@@_available_keys_str
10514 }
10515 {
10516     The~available~keys~are~(in~alphabetic~order):~
10517     &~in~blocks,~
10518     ampersand~in~blocks,~
10519     b,~
10520     baseline,~
10521     c,~
10522     cell-space-bottom-limit,~
10523     cell-space-limits,~
10524     cell-space-top-limit,~

```

```

10525     code-after,~
10526     code-for-first-col,~
10527     code-for-first-row,~
10528     code-for-last-col,~
10529     code-for-last-row,~
10530     columns-type,~
10531     columns-width,~
10532     corners,~
10533     create-extra-nodes,~
10534     create-medium-nodes,~
10535     create-large-nodes,~
10536     extra-left-margin,~
10537     extra-right-margin,~
10538     first-col,~
10539     first-row,~
10540     hlines,~
10541     hvlines,~
10542     hvlines-except-borders,~
10543     l,~
10544     last-col,~
10545     last-row,~
10546     left-margin,~
10547     light-syntax,~
10548     light-syntax-expanded,~
10549     name,~
10550     no-cell-nodes,~
10551     nullify-dots,~
10552     pgf-node-code,~
10553     r,~
10554     renew-dots,~
10555     respect-arraystretch,~
10556     right-margin,~
10557     rounded-corners,~
10558     rules~(with~the~subkeys~'color'~and~'width'),~
10559     small,~
10560     t,~
10561     vlines,~
10562     xdots/color,~
10563     xdots/shorten-start,~
10564     xdots/shorten-end,~
10565     xdots/shorten-and~
10566     xdots/line-style.
10567 }
10568 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10569 {
10570   Unknown~key.\\
10571   The~key~' \l_keys_key_str 'is~unknown~for~the~environment~
10572   \{NiceTabular\}. \\
10573   That~key~will~be~ignored. \\
10574   \c_@@_available_keys_str
10575 }
10576 {
10577   The~available~keys~are~(in~alphabetic~order):~
10578   &~in~blocks,~
10579   ampersand~in~blocks,~
10580   b,~
10581   baseline,~
10582   c,~
10583   caption,~
10584   cell-space-bottom-limit,~
10585   cell-space-limits,~
10586   cell-space-top-limit,~
10587   code-after,~

```

```

10588 code-for-first-col,~
10589 code-for-first-row,~
10590 code-for-last-col,~
10591 code-for-last-row,~
10592 columns-width,~
10593 corners,~
10594 custom-line,~
10595 create-extra-nodes,~
10596 create-medium-nodes,~
10597 create-large-nodes,~
10598 extra-left-margin,~
10599 extra-right-margin,~
10600 first-col,~
10601 first-row,~
10602 hlines,~
10603 hvlines,~
10604 hvlines-except-borders,~
10605 label,~
10606 last-col,~
10607 last-row,~
10608 left-margin,~
10609 light-syntax,~
10610 light-syntax-expanded,~
10611 name,~
10612 no-cell-nodes,~
10613 notes~(several~subkeys),~
10614 nullify-dots,~
10615 pgf-node-code,~
10616 renew-dots,~
10617 respect-arraystretch,~
10618 right-margin,~
10619 rounded-corners,~
10620 rules~(with~the~subkeys~'color'~and~'width'),~
10621 short-caption,~
10622 t,~
10623 tabularnote,~
10624 vlines,~
10625 xdots/color,~
10626 xdots/shorten-start,~
10627 xdots/shorten-end,~
10628 xdots/shorten-and~
10629 xdots/line-style.
10630 }

10631 \@@_msg_new:nnn { Duplicate~name }
10632 {
10633   Duplicate~name.\\
10634   The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
10635   the~same~environment~name~twice.~You~can~go~on,~but,~
10636   maybe,~you~will~have~incorrect~results~especially~
10637   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10638   message~again,~use~the~key~'allow-duplicate-names'~in~
10639   ' \token_to_str:N \NiceMatrixOptions '~.\\
10640   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10641     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10642 }
10643 {
10644   The~names~already~defined~in~this~document~are::~
10645   \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10646 }

10647 \@@_msg_new:nn { Option~auto~for~columns-width }
10648 {
10649   Erroneous~use.\\
10650   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~

```

```

10651     That~key~will~be~ignored.
10652 }
10653 \@@_msg_new:nn { NiceTabularX~without~X }
10654 {
10655     NiceTabularX~without~X.\\
10656     You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
10657     However,~you~can~go~on.
10658 }
10659 \@@_msg_new:nn { Preamble~forgotten }
10660 {
10661     Preamble~forgotten.\\
10662     You~have~probably~forgotten~the~preamble~of~your~
10663     \@@_full_name_env: . \\
10664     This~error~is~fatal.
10665 }
10666 \@@_msg_new:nn { Invalid~col~number }
10667 {
10668     Invalid~column~number.\\
10669     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10670     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10671 }
10672 \@@_msg_new:nn { Invalid~row~number }
10673 {
10674     Invalid~row~number.\\
10675     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
10676     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10677 }
10678 \@@_define_com:NNN p ( )
10679 \@@_define_com:NNN b [ ]
10680 \@@_define_com:NNN v | |
10681 \@@_define_com:NNN V \| \|
10682 \@@_define_com:NNN B \{ \}

```

# Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	24
7	The options	25
8	Important code used by <code>{NiceArrayWithDelims}</code>	36
9	The <code>\CodeBefore</code>	50
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	75
13	The environment <code>{NiceMatrix}</code> and its variants	93
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
15	After the construction of the array	95
16	We draw the dotted lines	102
17	The actual instructions for drawing the dotted lines with Tikz	116
18	User commands available in the new environments	121
19	The command <code>\line</code> accessible in code-after	128
20	The command <code>\RowStyle</code>	129
21	Colors of cells, rows and columns	132
22	The vertical and horizontal rules	144
23	The empty corners	161
24	The environment <code>{NiceMatrixBlock}</code>	163
25	The extra nodes	164
26	The blocks	169
27	How to draw the dotted lines transparently	193
28	Automatic arrays	194
29	The redefinition of the command <code>\dotfill</code>	195
30	The command <code>\diagbox</code>	195

31	The keyword <code>\CodeAfter</code>	197
32	The delimiters in the preamble	197
33	The command <code>\SubMatrix</code>	199
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	208
35	The commands <code>HBrace</code> et <code>VBrace</code>	211
36	The command <code>TikzEveryCell</code>	213
37	The command <code>\ShowCellNames</code>	215
38	We process the options at package loading	216
39	About the package underscore	218
40	Error messages of the package	218