

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

October 23, 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 translation: `nicematrix-french.pdf`).

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

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

---

\*This document corresponds to the version 7.4 of `nicematrix`, at the date of 2025/10/23.

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

```

20 \RequirePackage { amsmath }

21 \RequirePackage { array }

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

```

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

```

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

```

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

```

36 \cs_new_protected:Npn \@@_error_or_warning:n
37 {
38   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
39     { \@@_warning:n }
40     { \@@_error:n }
41 }

```

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".

```

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

48 \@@_msg_new:nn { mdwtab-loaded }
49 {
50   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
51   This~error~is~fatal.
52 }

53 \hook_gput_code:nnn { begindocument / end } { . }
54 { \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`).

```

55 \cs_new_protected:Npn \@@_collect_options:n #1
56 {
57   \peek_meaning:NTF [
58     { \@@_collect_options:nw { #1 } }
59     { #1 { } }
60   }

```

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

```

61 \NewDocumentCommand \@@_collect_options:nw { m r[] }
62 { \@@_collect_options:nn { #1 } { #2 } }
63
64 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
65 {
66   \peek_meaning:NTF [
67     { \@@_collect_options:nnw { #1 } { #2 } }
68     { #1 { #2 } }
69   }
70
71 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
72 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

## 3 Technical definitions

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

```

73 \tl_const:Nn \c_@@_c_tl { c }
74 \tl_const:Nn \c_@@_l_tl { l }
75 \tl_const:Nn \c_@@_r_tl { r }
76 \tl_const:Nn \c_@@_all_tl { all }
77 \tl_const:Nn \c_@@_dot_tl { . }
78 \str_const:Nn \c_@@_r_str { r }
79 \str_const:Nn \c_@@_c_str { c }
80 \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).

```

81 \tl_new:N \l_@@_argspec_tl

```

```

82 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
83 \cs_generate_variant:Nn \str_set:Nn { N o }
84 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
85 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
86 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
87 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
88 \cs_generate_variant:Nn \dim_min:nn { v }
89 \cs_generate_variant:Nn \dim_max:nn { v }

90 \hook_gput_code:nnn { begindocument } { . }
91 {
92   \IfPackageLoadedTF { tikz }
93   {

```

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.

```

94   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
95   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
96 }
97 {
98   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
99   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
100 }
101 }

```

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`).

```

102 \IfClassLoadedTF { revtex4-1 }
103 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
104 {
105   \IfClassLoadedTF { revtex4-2 }
106   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
107   {

```

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

```

108   \cs_if_exist:NT \rvtx@ifformat@geq
109   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
110   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
111 }
112 }

```

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.

```

113 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
114 {
115   \iow_now:Nn \@mainaux
116   {
117     \ExplSyntaxOn
118     \cs_if_free:NT \pgfsyspdfmark
119     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
120     \ExplSyntaxOff
121   }
122   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
123 }

```

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.

```

124 \ProvideDocumentCommand \iddots { }
125 {
126   \mathinner
127   {
128     \mkern 1 mu
129     \box_move_up:nn { 1 pt } { \hbox { . } }
130     \mkern 2 mu
131     \box_move_up:nn { 4 pt } { \hbox { . } }
132     \mkern 2 mu
133     \box_move_up:nn { 7 pt }
134     { \vbox:n { \kern 7 pt \hbox { . } } }
135     \mkern 1 mu
136   }
137 }

```

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.

```

138 \hook_gput_code:nnn { begindocument } { . }
139 {
140   \IfPackageLoadedT { booktabs }
141   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
142 }
143 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
144 {
145   \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`).

```

146   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
147   {
148     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
149     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
150   }
151 }

```

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

```

152 \hook_gput_code:nnn { begindocument } { . }
153 {
154   \cs_set_protected:Npe @@_everycr:
155   {
156     \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
157     { \noalign { \@@_in_everycr: } }
158   }
159   \IfPackageLoadedTF { colortbl }
160   {
161     \cs_set_eq:NN @@_old_cellcolor: \cellcolor
162     \cs_set_eq:NN @@_old_rowcolor: \rowcolor
163     \cs_new_protected:Npn @@_revert_colortbl:
164     {
165       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
166       {
167         \cs_set_eq:NN \cellcolor @@_old_cellcolor:
168         \cs_set_eq:NN \rowcolor @@_old_rowcolor:

```

```

169         }
170     }

```

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).

```

171     \cs_new_protected:Npn \@@_replace_columncolor:
172     {
173         \tl_replace_all:Nnn \g_@@_array_preamble_tl
174         { \columncolor }
175         { \@@_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.

```

176     }
177 }
178 {
179     \cs_new_protected:Npn \@@_revert_colortbl: { }
180     \cs_new_protected:Npn \@@_replace_columncolor:
181     { \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.

```

182     \def \CT@arc@ { }
183     \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
184     \def \CT@arc@ #1 #2
185     {
186         \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
187         { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
188     }

```

Idem for `\CT@drs@`.

```

189     \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
190     \def \CT@drs@ #1 #2
191     {
192         \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
193         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
194     }
195     \def \hline
196     {
197         \noalign { \ifnum 0 = ` } \fi
198         \cs_set_eq:NN \hskip \vskip
199         \cs_set_eq:NN \vrule \hrule
200         \cs_set_eq:NN \@width \@height
201         { \CT@arc@ \vline }
202         \futurelet \reserved@a
203         \@xhline
204     }
205 }
206 }

```

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.

```

207 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
208 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
209 {
210     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
211     \int_compare:nNnT { #1 } > { \c_one_int }
212     { \multispan { \int_eval:n { #1 - 1 } } & }
213     \multispan { \int_eval:n { #2 - #1 + 1 } }
214     {
215         \CT@arc@
216         \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>

```
217     \skip_horizontal:N \c_zero_dim
218 }
```

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`.

```
219     \everycr { }
220     \cr
221     \noalign { \skip_vertical:n { - \arrayrulewidth } }
222 }
```

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.

```
223 \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`.

```
224 { \@@_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*.

```
225 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
226 \cs_generate_variant:Nn \@@_cline_i:nn { e }
227 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
228 {
229     \tl_if_empty:nTF { #3 }
230     { \@@_cline_iii:w #1|#2-#2 \q_stop }
231     { \@@_cline_ii:w #1|#2-#3 \q_stop }
232 }
233 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
234 { \@@_cline_iii:w #1|#2-#3 \q_stop }
235 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
236 {
```

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).

```
237     \int_compare:nNnT { #1 } < { #2 }
238     { \multispan { \int_eval:n { #2 - #1 } } & }
239     \multispan { \int_eval:n { #3 - #2 + 1 } }
240     {
241         \CT@arc@
242         \leaders \hrule \@height \arrayrulewidth \hfill
243         \skip_horizontal:N \c_zero_dim
244     }
```

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

```
245     \peek_meaning_remove_ignore_spaces:NNTF \cline
246     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
247     { \everycr { } \cr }
248 }
```

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

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

---

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

```

250 \cs_new_protected:Npn \@@_set_CTarc:n #1
251 {
252   \tl_if_blank:nF { #1 }
253   {
254     \tl_if_head_eq_meaning:nNTF { #1 } [
255       { \def \CT@arc@ { \color #1 } }
256       { \def \CT@arc@ { \color { #1 } } }
257     ]
258   }
259 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

260 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
261 {
262   \tl_if_head_eq_meaning:nNTF { #1 } [
263     { \def \CT@drsc@ { \color #1 } }
264     { \def \CT@drsc@ { \color { #1 } } }
265   ]

```

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

```

266 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
267 {
268   \tl_if_head_eq_meaning:nNTF { #2 } [
269     { #1 #2 }
270     { #1 { #2 } }
271   ]
272 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

273 \cs_new_protected:Npn \@@_color:n #1
274 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
275 \cs_generate_variant:Nn \@@_color:n { o }

```

```

276 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
277 {
278   \tl_set_rescan:Nno
279   #1
280   {
281     \char_set_catcode_other:N >
282     \char_set_catcode_other:N <
283   }
284   #1
285 }

```

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

```

286 \dim_new:N \l_@@_tmpc_dim
287 \dim_new:N \l_@@_tmpd_dim
288 \tl_new:N \l_@@_tmpc_tl
289 \tl_new:N \l_@@_tmpd_tl
290 \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.

```
291 \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).

```
292 \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.

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

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

```
294 \box_new:N \l_@@_the_array_box
```

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

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

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

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

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

```
298 \bool_new:N \g_@@_delims_bool
299 \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.

```
300 \bool_new:N \l_@@_preamble_bool
301 \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.

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

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

```
303 \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.

```
304 \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).

```
305 \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.).

```
306 \dim_new:N \l_@@_col_width_dim
307 \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.

```
308 \int_new:N \g_@@_row_total_int
309 \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).

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

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

```
311 \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.

```
312 \tl_new:N \l_@@_hpos_cell_tl
313 \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`.

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

Idem for the mono-row blocks.

```
315 \dim_new:N \g_@@_blocks_ht_dim
316 \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`).

```
317 \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`.

```
318 \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.

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

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

```
320 \bool_new:N \l_@@_notes_detect_duplicates_bool
321 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```

322 \bool_new:N \l_@@_initial_open_bool
323 \bool_new:N \l_@@_final_open_bool
324 \bool_new:N \l_@@_Vbrace_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.

```

325 \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).

```

326 \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).

```

327 \tl_new:N \l_@@_rule_color_tl

```

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

```

328 \bool_new:N \g_@@_rotate_bool

```

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

```

329 \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.

```

330 \bool_new:N \l_@@_X_bool

```

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

```

331 \bool_new:N \l_@@_V_of_X_bool

```

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

```

332 \bool_new:N \g_@@_V_of_X_bool

```

```

333 \bool_new:N \g_@@_caption_finished_bool

```

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

```

334 \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 }`).

```

335 \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.

```

336 \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.

```

337 \seq_new:N \g_@@_size_seq

```

```

338 \tl_new:N \g_@@_left_delim_tl

```

```

339 \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}`).

```
340 \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`).

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

For `\multicolumn`.

```
342 \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`.

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

```
344 \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 `..`.

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

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

```
347 \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`).

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

```
349 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
350 {
```

```
351   \if_mode_math: \else:
```

```
352     \@@_fatal:n { Outside~math~mode }
```

```
353   \fi:
```

```
354 }
```

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analysis of the preamble of the array.

```
355 \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”.

```
356 \colorlet { nicematrix-last-col } { . }
```

```
357 \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`).

```
358 \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*.

```
359 \str_new:N \g_@@_com_or_env_str
```

```
360 \str_gset:Nn \g_@@_com_or_env_str { environment }
```

```
361 \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)`.

```

362 \cs_new:Npn \@@_full_name_env:
363 {
364   \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
365   { command \space \c_backslash_str \g_@@_name_env_str }
366   { environment \space \{ \g_@@_name_env_str \} }
367 }

368 \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.

```

369 \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.

```

370 \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.

```

371 \tl_new:N \g_@@_pre_code_before_tl
372 \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.

```

373 \tl_new:N \g_@@_pre_code_after_tl
374 \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.

```

375 \bool_new:N \l_@@_in_code_after_bool

```

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

```

376 \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.

```

377 \int_new:N \l_@@_old_iRow_int
378 \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).

```

379 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

380 \tl_new:N \l_@@_rules_color_tl

```

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

```
381 \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.

```
382 \bool_new:N \l_@@_X_columns_aux_bool
383 \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.

```
384 \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).

```
385 \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).

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

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

```
388 \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).

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

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

```
390 \dim_new:N \l_@@_x_initial_dim
391 \dim_new:N \l_@@_y_initial_dim
392 \dim_new:N \l_@@_x_final_dim
393 \dim_new:N \l_@@_y_final_dim
```

```
394 \dim_new:N \g_@@_dp_row_zero_dim
395 \dim_new:N \g_@@_ht_row_zero_dim
396 \dim_new:N \g_@@_ht_row_one_dim
397 \dim_new:N \g_@@_dp_ante_last_row_dim
398 \dim_new:N \g_@@_ht_last_row_dim
399 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
400 \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”.

```
401 \dim_new:N \g_@@_width_last_col_dim
402 \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.

```
403 \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!).

```
404 \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.

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

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocks_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}`.

```
406 \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.

```
407 \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`).

```
408 \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`).

```
409 \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.

```
410 \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).

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

```
412 \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.

```
413 \int_new:N \g_@@_ddots_int
414 \int_new:N \g_@@_iddots_int
```

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

```
415 \dim_new:N \g_@@_delta_x_one_dim
416 \dim_new:N \g_@@_delta_y_one_dim
417 \dim_new:N \g_@@_delta_x_two_dim
418 \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).

```
419 \int_new:N \l_@@_row_min_int
420 \int_new:N \l_@@_row_max_int
421 \int_new:N \l_@@_col_min_int
422 \int_new:N \l_@@_col_max_int

423 \int_new:N \l_@@_initial_i_int
424 \int_new:N \l_@@_initial_j_int
425 \int_new:N \l_@@_final_i_int
426 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
427 \int_new:N \l_@@_start_int
428 \int_set_eq:NN \l_@@_start_int \c_one_int
429 \int_new:N \l_@@_end_int
430 \int_new:N \l_@@_local_start_int
431 \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.

```
432 \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).

```
433 \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`.

```
434 \tl_new:N \l_@@_fill_tl
435 \tl_new:N \l_@@_opacity_tl
436 \tl_new:N \l_@@_draw_tl
437 \seq_new:N \l_@@_tikz_seq
438 \clist_new:N \l_@@_borders_clist
439 \dim_new:N \l_@@_rounded_corners_dim
```

---

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



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

The 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.

```
440 \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`.

```
441 \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.

```
442 \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.

```
443 \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`).

```
444 \str_new:N \l_@@_hpos_block_str
445 \str_set:Nn \l_@@_hpos_block_str { c }
446 \bool_new:N \l_@@_hpos_of_block_cap_bool
447 \bool_new:N \l_@@_p_block_bool
```

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

```
448 \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).

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

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

```
450 \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).

```
451 \bool_new:N \l_@@_vlines_block_bool
452 \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.

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

454 \dim_new:N \l_@@_submatrix_extra_height_dim
455 \dim_new:N \l_@@_submatrix_left_xshift_dim
456 \dim_new:N \l_@@_submatrix_right_xshift_dim
457 \clist_new:N \l_@@_hlines_clist
458 \clist_new:N \l_@@_vlines_clist
459 \clist_new:N \l_@@_submatrix_hlines_clist
460 \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}`).

```
461 \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.

```
462 \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`).

```
463 \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.

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

```
466 \int_new:N \l_@@_first_col_int
467 \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).

```
468 \int_new:N \l_@@_last_row_int
469 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
471 \bool_new:N \l_@@_last_col_without_value_bool
```

---

<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.

- **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  $0$ .

```
472 \int_new:N \l_@@_last_col_int
473 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
474 \bool_new:N \g_@@_last_col_found_bool
```

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

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

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

## Some utilities

```
476 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
477 {
```

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

```
478 \def \l_tmpa_tl { #1 }
479 \def \l_tmpb_tl { #2 }
480 }
```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers. The second argument is `\c@iRow` or `\c@jCol`.

```
481 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
482 {
483   \clist_if_in:NnF #1 { all }
484   {
485     \clist_clear:N \l_tmpa_clist
486     \clist_map_inline:Nn #1
487     {
488       \tl_if_head_eq_meaning:nNTF { ##1 } -
489       {
```

If we have yet the number of columns or the number of columns (because they have been computed during a previous run and written on the `aux` file), we can compute the actual position of the rule with a negative position.

```
490       \int_if_zero:nF { #2 }
491       {
492         \clist_put_right:Ne \l_tmpa_clist
493         { \int_eval:n { #2 + (##1) + 1 } }
494       }
495     }
496   }
```

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

```

497         \tl_if_in:nnTF { ##1 } { - }
498         { \@@_cut_on_hyphen:w ##1 \q_stop }
499         {

```

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

```

500         \def \l_tmpa_tl { ##1 }
501         \def \l_tmpb_tl { ##1 }
502     }
503     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
504     { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
505 }
506 }
507 \tl_set_eq:NN #1 \l_tmpa_clist
508 }
509 }

```

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row;
- when the special character “:” is used in order to put the label of a so-called “dotted line” on the line, a margin of `\c_@@_innersep_middle_dim` will be added around the label.

```

510 \hook_gput_code:nnn { begindocument } { . }
511 {
512     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
513     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
514 }

```

## 5 The command `\tabularnote`

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

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

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

---

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

- 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.

```
515 \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`.

```
516 \int_new:N \g_@@_tabularnote_int
517 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
518 \seq_new:N \g_@@_notes_seq
519 \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.

```
520 \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.

```
521 \seq_new:N \l_@@_notes_labels_seq
522 \newcounter { nicematrix_draft }
523 \cs_new_protected:Npn \@@_notes_format:n #1
524 {
525   \setcounter { nicematrix_draft } { #1 }
526   \@@_notes_style:n { nicematrix_draft }
527 }
```

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

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

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

```
529 \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`.

```
530 \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`.

```
531 \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`).

```
532 \hook_gput_code:nnn { begindocument } { . }
533 {
534   \IfPackageLoadedTF { enumitem }
535   {
```

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.

```

536     \newlist { tabularnotes } { enumerate } { 1 }
537     \setlist [ tabularnotes ]
538     {
539         topsep = \c_zero_dim ,
540         noitemsep ,
541         leftmargin = * ,
542         align = left ,
543         labelsep = \c_zero_dim ,
544         label =
545             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
546     }
547     \newlist { tabularnotes* } { enumerate* } { 1 }
548     \setlist [ tabularnotes* ]
549     {
550         afterlabel = \nobreak ,
551         itemjoin = \quad ,
552         label =
553             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
554     }

```

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`.

```

555     \NewDocumentCommand \tabularnote { o m }
556     {
557         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
558         {
559             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
560             { \@@_error:n { tabularnote~forbidden } }
561             {
562                 \bool_if:NTF \l_@@_in_caption_bool
563                 \@@_tabularnote_caption:nn
564                 \@@_tabularnote:nn
565                 { #1 } { #2 }
566             }
567         }
568     }
569 }
570 {
571     \NewDocumentCommand \tabularnote { o m }
572     { \@@_err_enumitem_not_loaded: }
573 }
574 }

575 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
576 {
577     \@@_error_or_warning:n { enumitem~not~loaded }
578     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
579 }

580 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
581 { \tl_if_novalue:nT { #1 } { #3 } }

```

For the version in normal conditions, that is to say not in the caption. `#1` is the optional argument of `\tabularnote` (maybe equal to the special marker expressed by `\NoValue`) and `#2` is the mandatory argument of `\tabularnote`.

```

582 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
583 {

```

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).

```

584     \int_zero:N \l_tmpa_int
585     \bool_if:NT \l_@@_notes_detect_duplicates_bool
586     {

```

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

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

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

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

```

587     \int_zero:N \l_tmpb_int
588     \seq_map_indexed_inline:Nn \g_@@_notes_seq
589     {
590         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
591         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 } }
592         {
593             \tl_if_novalue:nTF { #1 }
594             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
595             { \int_set:Nn \l_tmpa_int { ##1 } }
596             \seq_map_break:
597         }
598     }
599     \int_if_zero:nF { \l_tmpa_int }
600     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
601 }
602 \int_if_zero:nT { \l_tmpa_int }
603 {
604     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
605     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
606 }
607 \seq_put_right:Ne \l_@@_notes_labels_seq
608 {
609     \tl_if_novalue:nTF { #1 }
610     {
611         \@@_notes_format:n
612         {
613             \int_eval:n
614             {
615                 \int_if_zero:nTF { \l_tmpa_int }
616                 { \c@tabularnote }
617                 { \l_tmpa_int }
618             }
619         }
620     }
621     { #1 }
622 }
623 \peek_meaning:NF \tabularnote
624 {

```

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`.

```

625     \hbox_set:Nn \l_tmpa_box
626     {

```

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

```

627         \@@_notes_label_in_tabular:n
628         {
629             \seq_use:Nnnn
630             \l_@@_notes_labels_seq { , } { , } { , }
631         }
632     }

```

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

```

633     \int_gdecr:N \c@tabularnote
634     \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.

```

635     \int_gincr:N \g_@@_tabularnote_int
636     \refstepcounter { tabularnote }
637     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
638     { \int_gincr:N \c@tabularnote }
639     \seq_clear:N \l_@@_notes_labels_seq
640     \bool_lazy_or:nnTF
641     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
642     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
643     {
644         \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).

```

645         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
646     }
647     { \box_use:N \l_tmpa_box }
648 }
649 }

```

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`.

```

650 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
651 {
652     \bool_if:NTF \g_@@_caption_finished_bool
653     {
654         \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
655         { \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`!

```

656         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
657         { \@@_error:n { Identical~notes~in~caption } }
658     }
659     {

```

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

```

660         \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
661         {

```

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.

```

662         \bool_gset_true:N \g_@@_caption_finished_bool

```



```

663         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
664         \int_gzero:N \c@tabularnote
665     }
666     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
667 }

```

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!

```

668     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
669     \seq_put_right:Ne \l_@@_notes_labels_seq
670     {
671         \tl_if_novalue:nTF { #1 }
672         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
673         { #1 }
674     }
675     \peek_meaning:NF \tabularnote
676     {
677         \@@_notes_label_in_tabular:n
678         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
679         \seq_clear:N \l_@@_notes_labels_seq
680     }
681 }

682 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
683 { \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.

```

684 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
685 {
686     \begin { pgfscope }
687     \pgfset
688     {
689         inner~sep = \c_zero_dim ,
690         minimum~size = \c_zero_dim
691     }
692     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
693     \pgfnode
694     { rectangle }
695     { center }
696     {
697         \vbox_to_ht:nn
698         { \dim_abs:n { #5 - #3 } }
699         {
700             \vfill
701             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { } { }
702         }
703     }
704     { #1 }
705     { }
706     \end { pgfscope }
707 }

```

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.

```

708 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
709 {

```

```

710 \begin { pgfscope }
711 \pgfset
712 {
713     inner~sep = \c_zero_dim ,
714     minimum~size = \c_zero_dim
715 }
716 \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
717 \pgfpointdiff { #3 } { #2 }
718 \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
719 \pgfnode
720 { rectangle }
721 { center }
722 {
723     \vbox_to_ht:nn
724     { \dim_abs:n \l_tmpb_dim }
725     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
726 }
727 { #1 }
728 { }
729 \end { pgfscope }
730 }

```

## 7 The options

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

```

731 \tl_new:N \l_@@_caption_tl
732 \tl_new:N \l_@@_short_caption_tl
733 \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).

```

734 \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`.

```

735 \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`).

```

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

```

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

```

738 \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.

```

739 \dim_new:N \l_@@_xdots_inter_dim
740 \hook_gput_code:nnn { begindocument } { . }
741 { \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).

```

742 \dim_new:N \l_@@_xdots_shorten_start_dim
743 \dim_new:N \l_@@_xdots_shorten_end_dim
744 \hook_gput_code:nnn { begindocument } { . }
745 {
746   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
747   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
748 }

```

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.

```

749 \dim_new:N \l_@@_xdots_radius_dim
750 \hook_gput_code:nnn { begindocument } { . }
751 { \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.

```

752 \tl_new:N \l_@@_xdots_line_style_tl
753 \tl_const:Nn \c_@@_standard_tl { standard }
754 \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`.

```

755 \bool_new:N \l_@@_light_syntax_bool
756 \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).

```

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

```

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

```

759 \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`).

```

760 \bool_new:N \l_@@_exterior_arraycolsep_bool

```

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

```

761 \bool_new:N \l_@@_parallelize_diags_bool
762 \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.

```

763 \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.).

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

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

```
765 \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).

```
766 \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!

```
767 \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.

```
768 \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”.

```
769 \bool_new:N \l_@@_medium_nodes_bool
```

```
770 \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).

```
771 \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`).

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

```
773 \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`.

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

```
775 \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.

```
776 \tl_new:N \l_@@_end_of_row_tl
```

```
777 \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 “:”.

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

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

```
779 \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 function 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.

```

780 \bool_new:N \l_@@_delimiters_max_width_bool

781 \keys_define:nn { nicematrix / xdots }
782 {
783   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
784   shorten-start .code:n =
785     \hook_gput_code:nnn { begindocument } { . }
786     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
787   shorten-end .code:n =
788     \hook_gput_code:nnn { begindocument } { . }
789     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
790   shorten-start .value_required:n = true ,
791   shorten-end .value_required:n = true ,
792   shorten .code:n =
793     \hook_gput_code:nnn { begindocument } { . }
794     {
795       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
796       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
797     } ,
798   shorten .value_required:n = true ,
799   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
800   horizontal-labels .default:n = true ,
801   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
802   horizontal-label .default:n = true ,
803   line-style .code:n =
804     {
805       \bool_lazy_or:nnTF
806       { \cs_if_exist_p:N \tikzpicture }
807       { \str_if_eq_p:nn { #1 } { standard } }
808       { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
809       { \@@_error:n { bad-option-for-line-style } }
810     } ,
811   line-style .value_required:n = true ,
812   color .tl_set:N = \l_@@_xdots_color_tl ,
813   color .value_required:n = true ,
814   radius .code:n =
815     \hook_gput_code:nnn { begindocument } { . }
816     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
817   radius .value_required:n = true ,
818   inter .code:n =
819     \hook_gput_code:nnn { begindocument } { . }
820     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
821   radius .value_required:n = true ,

```

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

```

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

```

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

```

825   draw-first .code:n = \prg_do_nothing: ,

```

```

826     unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
827 }

828 \keys_define:nn { nicematrix / rules }
829 {
830     color .tl_set:N = \l_@@_rules_color_tl ,
831     color .value_required:n = true ,
832     width .dim_set:N = \arrayrulewidth ,
833     width .value_required:n = true ,
834     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
835 }

836 \cs_new_protected:Npn \@@_err_key_color_inside:
837 {
838     \@@_error_or_warning:n { key-color-inside }
839     \cs_gset:Npn \@@_err_key_color_inside: { }
840 }

```

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

```

841 \keys_define:nn { nicematrix / Global }
842 {
843     color-inside .code:n = \@@_err_key_color_inside: ,
844     colortbl-like .code:n = \@@_err_key_color_inside: ,
845     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
846     ampersand-in-blocks .default:n = true ,
847     &-in-blocks .meta:n = ampersand-in-blocks ,
848     no-cell-nodes .code:n =
849         \bool_set_true:N \l_@@_no_cell_nodes_bool
850         \cs_set_protected:Npn \@@_node_cell:
851             { \set@color \box_use_drop:N \l_@@_cell_box } ,
852     no-cell-nodes .value_forbidden:n = true ,
853     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
854     rounded-corners .default:n = 4 pt ,
855     custom-line .code:n = \@@_custom_line:n { #1 } ,
856     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
857     rules .value_required:n = true ,
858     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
859     standard-cline .default:n = true ,
860     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
861     cell-space-top-limit .value_required:n = true ,
862     cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
863     cell-space-bottom-limit .value_required:n = true ,
864     cell-space-limits .meta:n =
865         {
866             cell-space-top-limit = #1 ,
867             cell-space-bottom-limit = #1 ,
868         } ,
869     cell-space-limits .value_required:n = true ,
870     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
871     light-syntax .code:n =
872         \bool_set_true:N \l_@@_light_syntax_bool
873         \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
874     light-syntax .value_forbidden:n = true ,
875     light-syntax-expanded .code:n =
876         \bool_set_true:N \l_@@_light_syntax_bool
877         \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
878     light-syntax-expanded .value_forbidden:n = true ,
879     end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
880     end-of-row .value_required:n = true ,
881     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
882     first-row .code:n = \int_zero:N \l_@@_first_row_int ,

```

```

883 last-row .int_set:N = \l_@@_last_row_int ,
884 last-row .default:n = -1 ,
885 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
886 code-for-first-col .value_required:n = true ,
887 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
888 code-for-last-col .value_required:n = true ,
889 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
890 code-for-first-row .value_required:n = true ,
891 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
892 code-for-last-row .value_required:n = true ,
893 hlines .clist_set:N = \l_@@_hlines_clist ,
894 vlines .clist_set:N = \l_@@_vlines_clist ,
895 hlines .default:n = all ,
896 vlines .default:n = all ,
897 vlines-in-sub-matrix .code:n =
898 {
899   \tl_if_single_token:nTF { #1 }
900   {
901     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
902     { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

903   { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
904   }
905   { \@@_error:n { One~letter~allowed } }
906   } ,
907 vlines-in-sub-matrix .value_required:n = true ,
908 hvlines .code:n =
909 {
910   \bool_set_true:N \l_@@_hvlines_bool
911   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
912   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
913   } ,
914 hvlines .value_forbidden:n = true ,
915 hvlines-except-borders .code:n =
916 {
917   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
918   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
919   \bool_set_true:N \l_@@_hvlines_bool
920   \bool_set_true:N \l_@@_except_borders_bool
921   } ,
922 hvlines-except-borders .value_forbidden:n = true ,
923 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.

```

924 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
925 renew-dots .value_forbidden:n = true ,
926 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
927 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
928 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
929 create-extra-nodes .meta:n =
930 { create-medium-nodes , create-large-nodes } ,
931 left-margin .dim_set:N = \l_@@_left_margin_dim ,
932 left-margin .default:n = \arraycolsep ,
933 right-margin .dim_set:N = \l_@@_right_margin_dim ,
934 right-margin .default:n = \arraycolsep ,
935 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
936 margin .default:n = \arraycolsep ,
937 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
938 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
939 extra-margin .meta:n =
940 { extra-left-margin = #1 , extra-right-margin = #1 } ,

```

```

941     extra-margin .value_required:n = true ,
942     respect-arraystretch .code:n =
943       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
944     respect-arraystretch .value_forbidden:n = true ,
945     pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
946     pgf-node-code .value_required:n = true
947   }

```

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

```

948 \keys_define:nn { nicematrix / environments }
949 {
950   corners .clist_set:N = \l_@@_corners_clist ,
951   corners .default:n = { NW , SW , NE , SE } ,
952   code-before .code:n =
953     {
954       \tl_if_empty:nF { #1 }
955       {
956         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
957         \bool_set_true:N \l_@@_code_before_bool
958       }
959     } ,
960   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}`.

```

961   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
962   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
963   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
964   baseline .tl_set:N = \l_@@_baseline_tl ,
965   baseline .value_required:n = true ,
966   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)`.

```

967   \str_if_eq:eeTF { #1 } { auto }
968   { \bool_set_true:N \l_@@_auto_columns_width_bool }
969   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
970   columns-width .value_required:n = true ,
971   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.

```

972   \legacy_if:nF { measuring@ }
973   {
974     \str_set:Ne \l_@@_name_str { #1 }
975     \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
976     { \@@_err_duplicate_names:n { #1 } }
977     { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
978   } ,
979   name .value_required:n = true ,
980   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
981   code-after .value_required:n = true ,
982 }

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

985 \keys_define:nn { nicematrix / notes }
986 {
987   para .bool_set:N = \l_@@_notes_para_bool ,
988   para .default:n = true ,
989   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
990   code-before .value_required:n = true ,

```



```

991 code-after .tl_set:N = \l_@@_notes_code_after_tl ,
992 code-after .value_required:n = true ,
993 bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
994 bottomrule .default:n = true ,
995 style .cs_set:Np = \@@_notes_style:n #1 ,
996 style .value_required:n = true ,
997 label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
998 label-in-tabular .value_required:n = true ,
999 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1000 label-in-list .value_required:n = true ,
1001 enumitem-keys .code:n =
1002 {
1003   \hook_gput_code:nnn { begindocument } { . }
1004   {
1005     \IfPackageLoadedT { enumitem }
1006       { \setlist* [ tabularnotes ] { #1 } }
1007   }
1008 } ,
1009 enumitem-keys .value_required:n = true ,
1010 enumitem-keys-para .code:n =
1011 {
1012   \hook_gput_code:nnn { begindocument } { . }
1013   {
1014     \IfPackageLoadedT { enumitem }
1015       { \setlist* [ tabularnotes* ] { #1 } }
1016   }
1017 } ,
1018 enumitem-keys-para .value_required:n = true ,
1019 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1020 detect-duplicates .default:n = true ,
1021 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1022 }
1023 \keys_define:nn { nicematrix / delimiters }
1024 {
1025   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1026   max-width .default:n = true ,
1027   color .tl_set:N = \l_@@_delimiters_color_tl ,
1028   color .value_required:n = true ,
1029 }

```

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

```

1030 \keys_define:nn { nicematrix }
1031 {
1032   NiceMatrixOptions .inherit:n =
1033     { nicematrix / Global } ,
1034   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1035   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1036   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1037   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1038   SubMatrix / rules .inherit:n = nicematrix / rules ,
1039   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1040   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1041   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1042   NiceMatrix .inherit:n =
1043     {
1044       nicematrix / Global ,
1045       nicematrix / environments ,
1046     } ,
1047   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1048   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1049   NiceTabular .inherit:n =

```

```

1050     {
1051         nicematrix / Global ,
1052         nicematrix / environments
1053     } ,
1054     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1055     NiceTabular / rules .inherit:n = nicematrix / rules ,
1056     NiceTabular / notes .inherit:n = nicematrix / notes ,
1057     NiceArray .inherit:n =
1058     {
1059         nicematrix / Global ,
1060         nicematrix / environments ,
1061     } ,
1062     NiceArray / xdots .inherit:n = nicematrix / xdots ,
1063     NiceArray / rules .inherit:n = nicematrix / rules ,
1064     pNiceArray .inherit:n =
1065     {
1066         nicematrix / Global ,
1067         nicematrix / environments ,
1068     } ,
1069     pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1070     pNiceArray / rules .inherit:n = nicematrix / rules ,
1071 }

```

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

```

1072 \keys_define:nn { nicematrix / NiceMatrixOptions }
1073 {
1074     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1075     delimiters / color .value_required:n = true ,
1076     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1077     delimiters / max-width .default:n = true ,
1078     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1079     delimiters .value_required:n = true ,
1080     width .dim_set:N = \l_@@_width_dim ,
1081     width .value_required:n = true ,
1082     last-col .code:n =
1083     \tl_if_empty:nF { #1 }
1084     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1085     \int_zero:N \l_@@_last_col_int ,
1086     small .bool_set:N = \l_@@_small_bool ,
1087     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.

```

1088     renew-matrix .code:n = \@@_renew_matrix: ,
1089     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}`.

```

1090     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.

```

1091     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)`.

```

1092     \str_if_eq:eeTF { #1 } { auto }
1093     { \@@_error:n { Option~auto~for~columns-width } }
1094     { \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.

```

1095   allow-duplicate-names .code:n =
1096     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1097   allow-duplicate-names .value_forbidden:n = true ,
1098   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1099   notes .value_required:n = true ,
1100   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1101   sub-matrix .value_required:n = true ,
1102   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1103   matrix / columns-type .value_required:n = true ,
1104   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1105   caption-above .default:n = true ,
1106   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1107 }

```

`\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.

```

1108 \NewDocumentCommand \NiceMatrixOptions { m }
1109 { \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.

```

1110 \keys_define:nn { nicematrix / NiceMatrix }
1111 {
1112   last-col .code:n = \tl_if_empty:nTF { #1 }
1113     {
1114       \bool_set_true:N \l_@@_last_col_without_value_bool
1115       \int_set:Nn \l_@@_last_col_int { -1 }
1116     }
1117     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1118   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1119   columns-type .value_required:n = true ,
1120   l .meta:n = { columns-type = l } ,
1121   r .meta:n = { columns-type = r } ,
1122   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1123   delimiters / color .value_required:n = true ,
1124   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1125   delimiters / max-width .default:n = true ,
1126   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1127   delimiters .value_required:n = true ,
1128   small .bool_set:N = \l_@@_small_bool ,
1129   small .value_forbidden:n = true ,
1130   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1131 }

```

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

```

1132 \keys_define:nn { nicematrix / NiceArray }
1133 {

```

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.

```

1134   small .bool_set:N = \l_@@_small_bool ,
1135   small .value_forbidden:n = true ,
1136   last-col .code:n = \tl_if_empty:nF { #1 }
1137     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1138     \int_zero:N \l_@@_last_col_int ,
1139   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1140   l .code:n = \@@_error:n { r-or-l-with-preamble } ,

```

```

1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1142   }
1143 \keys_define:nn { nicematrix / pNiceArray }
1144 {
1145   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1146   last-col .code:n = \tl_if_empty:nF { #1 }
1147     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1148     \int_zero:N \l_@@_last_col_int ,
1149   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1150   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1151   delimiters / color .value_required:n = true ,
1152   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1153   delimiters / max-width .default:n = true ,
1154   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1155   delimiters .value_required:n = true ,
1156   small .bool_set:N = \l_@@_small_bool ,
1157   small .value_forbidden:n = true ,
1158   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1159   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1160   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1161 }

```

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

```

1162 \keys_define:nn { nicematrix / NiceTabular }
1163 {

```

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.

```

1164   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1165   \bool_set_true:N \l_@@_width_used_bool ,
1166   width .value_required:n = true ,
1167   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1168   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1169   tabularnote .value_required:n = true ,
1170   caption .tl_set:N = \l_@@_caption_tl ,
1171   caption .value_required:n = true ,
1172   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1173   short-caption .value_required:n = true ,
1174   label .tl_set:N = \l_@@_label_tl ,
1175   label .value_required:n = true ,
1176   last-col .code:n = \tl_if_empty:nF { #1 }
1177     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1178     \int_zero:N \l_@@_last_col_int ,
1179   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1180   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1181   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1182 }

```

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

```

1183 \keys_define:nn { nicematrix / CodeAfter }
1184 {
1185   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1186   delimiters / color .value_required:n = true ,
1187   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1188   rules .value_required:n = true ,

```

```

1189     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1190     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1191     sub-matrix .value_required:n = true ,
1192     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1193 }

```

## 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}`).

```

1194 \cs_new_protected:Npn \@@_cell_begin:
1195 {

```

`\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.

```

1196     \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).

```

1197     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

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

```

1198     \cs_set_eq:NN \Hline \@@_Hline_in_cell:

```

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

```

1199     \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.

```

1200     \int_compare:nNnT { \c@jCol } = { \c_one_int }
1201     {
1202         \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1203         { \@@_begin_of_row: }
1204     }

```

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:`.

```

1205     \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1206     \@@_tuning_not_tabular_begin:
1207     \@@_tuning_first_row:
1208     \@@_tuning_last_row:
1209     \g_@@_row_style_tl
1210 }

```

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.

```

1211 \cs_new_protected:Npn \@@_tuning_first_row:
1212 {
1213   \if_int_compare:w \c@iRow = \c_zero_int
1214     \if_int_compare:w \c@jCol > \c_zero_int
1215       \l_@@_code_for_first_row_tl
1216       \xglobal \colorlet { nicematrix-first-row } { . }
1217     \fi:
1218   \fi:
1219 }

```

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

```

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

```

We will use a version a little more efficient.

```

1220 \cs_new_protected:Npn \@@_tuning_last_row:
1221 {
1222   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1223     \l_@@_code_for_last_row_tl
1224     \xglobal \colorlet { nicematrix-last-row } { . }
1225   \fi:
1226 }

```

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

```

1227 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1228 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1229 {
1230   \m@th
1231   \c_math_toggle_token

```

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

```

1232   \@@_tuning_key_small:
1233 }
1234 \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.

```

1235 \cs_new_protected:Npn \@@_begin_of_row:
1236 {
1237   \int_gincr:N \c@iRow
1238   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1239   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1240   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1241   \pgfpicture
1242   \pgfrememberpicturepositiononpagetrue
1243   \pgfcoordinate
1244     { \@@_env: - row - \int_use:N \c@iRow - base }
1245     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }

```

```

1246 \str_if_empty:NF \l_@@_name_str
1247 {
1248   \pgfnodealias
1249   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1250   { \@@_env: - row - \int_use:N \c@iRow - base }
1251 }
1252 \endpgfpicture
1253 }

```

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.

```

1254 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1255 {
1256   \int_if_zero:nTF { \c@iRow }
1257   {
1258     \dim_compare:nNnT
1259     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1260     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1261     \dim_compare:nNnT
1262     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1263     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1264   }
1265   {
1266     \int_compare:nNnT { \c@iRow } = { \c_one_int }
1267     {
1268       \dim_compare:nNnT
1269       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1270       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1271     }
1272   }
1273 }
1274 \cs_new_protected:Npn \@@_rotate_cell_box:
1275 {
1276   \box_rotate:Nn \l_@@_cell_box { 90 }
1277   \bool_if:NTF \g_@@_rotate_c_bool
1278   {
1279     \hbox_set:Nn \l_@@_cell_box
1280     {
1281       \m@th
1282       \c_math_toggle_token
1283       \vcenter { \box_use:N \l_@@_cell_box }
1284       \c_math_toggle_token
1285     }
1286   }
1287   {
1288     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1289     {
1290       \vbox_set_top:Nn \l_@@_cell_box
1291       {
1292         \vbox_to_zero:n { }
1293         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1294         \box_use:N \l_@@_cell_box
1295       }
1296     }
1297   }
1298   \bool_gset_false:N \g_@@_rotate_bool
1299   \bool_gset_false:N \g_@@_rotate_c_bool
1300 }

```

```

1301 \cs_new_protected:Npn \@@_adjust_size_box:
1302 {
1303   \dim_compare:nNt { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1304   {
1305     \box_set_wd:Nn \l_@@_cell_box
1306     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1307     \dim_gzero:N \g_@@_blocks_wd_dim
1308   }
1309   \dim_compare:nNt { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1310   {
1311     \box_set_dp:Nn \l_@@_cell_box
1312     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1313     \dim_gzero:N \g_@@_blocks_dp_dim
1314   }
1315   \dim_compare:nNt { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1316   {
1317     \box_set_ht:Nn \l_@@_cell_box
1318     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1319     \dim_gzero:N \g_@@_blocks_ht_dim
1320   }
1321 }
1322 \cs_new_protected:Npn \@@_cell_end:
1323 {

```

The following command is nullified in the tabulars.

```

1324   \@@_tuning_not_tabular_end:
1325   \hbox_set_end:
1326   \@@_cell_end_i:
1327 }
1328 \cs_new_protected:Npn \@@_cell_end_i:
1329 {

```

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.

```

1330   \g_@@_cell_after_hook_tl
1331   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1332   \@@_adjust_size_box:
1333   \box_set_ht:Nn \l_@@_cell_box
1334   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1335   \box_set_dp:Nn \l_@@_cell_box
1336   { \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”).

```

1337   \@@_update_max_cell_width:

```

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

```

1338   \@@_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.

```

1339 \bool_if:NTF \g_@@_empty_cell_bool
1340 { \box_use_drop:N \l_@@_cell_box }
1341 {
1342   \bool_if:NTF \g_@@_not_empty_cell_bool
1343   { \@@_print_node_cell: }
1344   {
1345     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1346     { \@@_print_node_cell: }
1347     { \box_use_drop:N \l_@@_cell_box }
1348   }
1349 }
1350 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1351 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1352 \bool_gset_false:N \g_@@_empty_cell_bool
1353 \bool_gset_false:N \g_@@_not_empty_cell_bool
1354 }

```

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

```

1355 \cs_new_protected:Npn \@@_update_max_cell_width:
1356 {
1357   \dim_gset:Nn \g_@@_max_cell_width_dim
1358   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1359 }

```

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`).

```

1360 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1361 {
1362   \@@_math_toggle:
1363   \hbox_set_end:
1364   \bool_if:NF \g_@@_rotate_bool
1365   {
1366     \hbox_set:Nn \l_@@_cell_box
1367     {
1368       \makebox [ \l_@@_col_width_dim ] [ s ]
1369       { \hbox_unpack_drop:N \l_@@_cell_box }
1370     }
1371   }
1372   \@@_cell_end_i:
1373 }

```

```

1374 \pgfset
1375 {
1376   nicematrix / cell-node /.style =
1377   {
1378     inner~sep = \c_zero_dim ,
1379     minimum~width = \c_zero_dim
1380   }
1381 }

```

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.

```

1382 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1383 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1384 {
1385   \use:c
1386   {
1387     __siunitx_table_align_
1388     \bool_if:NTF \l__siunitx_table_text_bool
1389       { \l__siunitx_table_align_text_tl }
1390       { \l__siunitx_table_align_number_tl }
1391     :n
1392   }
1393   { #1 }
1394 }

```

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!

```

1395 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1396 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1397 {
1398   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1399   \hbox:n
1400   {
1401     \pgfsys@markposition
1402     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1403   }
1404   #1
1405   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1406   \hbox:n
1407   {
1408     \pgfsys@markposition
1409     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1410   }
1411 }

1412 \cs_new_protected:Npn \@@_print_node_cell:
1413 {
1414   \socket_use:nn { nicematrix / siunitx-wrap }
1415   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1416 }

```

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

```

1417 \cs_new_protected:Npn \@@_node_cell:
1418 {
1419   \pgfpicture
1420   \pgfsetbaseline \c_zero_dim
1421   \pgfrememberpicturepositiononpagetrue
1422   \pgfset { nicematrix / cell-node }
1423   \pgfnode
1424   { rectangle }
1425   { base }
1426   {

```

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).

```

1427   \sys_if_engine_xetex:T { \set@color }
1428   \box_use:N \l_@@_cell_box
1429 }
1430 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1431 { \l_@@_pgf_node_code_tl }

```

```

1432 \str_if_empty:NF \l_@@_name_str
1433 {
1434   \pgfnodealias
1435     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1436     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1437 }
1438 \endpgfpicture
1439 }

```

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}

```

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

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

```

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

```

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

```

1440 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1441 {
1442   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1443   { g_@@_ #2 _ lines _ tl }
1444   {
1445     \use:c { @@ _ draw _ #2 : nnn }
1446     { \int_use:N \c@iRow }
1447     { \int_use:N \c@jCol }
1448     { \exp_not:n { #3 } }
1449   }
1450 }

1451 \cs_new_protected:Npn \@@_array:n
1452 {
1453   \dim_set:Nn \col@sep
1454   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1455   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1456   { \def \@halignto { } }
1457   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1458 \tabarray

```

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

```

1459 [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1460 ]
1461 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, it seems that RevTeX goes on with a redefinition of `array` which uses `\ialign`.

```
1462 \bool_if:NTF \c_@@_revtex_bool
1463 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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`.

```
1464 { \cs_set_eq:cN { \@@_old_ar@ialign: } \ar@ialign }
```

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

```
1465 \cs_new_protected:Npn \@@_create_row_node:
1466 {
1467   \int_compare:nNt { \c@iRow } > { \g_@@_last_row_node_int }
1468   {
1469     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1470     \@@_create_row_node_i:
1471   }
1472 }
1473 \cs_new_protected:Npn \@@_create_row_node_i:
1474 {
```

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

```
1475   \hbox
1476   {
1477     \bool_if:NT \l_@@_code_before_bool
1478     {
1479       \vtop
1480       {
1481         \skip_vertical:N 0.5\arrayrulewidth
1482         \pgfsys@markposition
1483         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1484         \skip_vertical:N -0.5\arrayrulewidth
1485       }
1486     }
1487     \pgfpicture
1488     \pgfrememberpicturepositiononpagetrue
1489     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1491     \str_if_empty:NF \l_@@_name_str
1492     {
1493       \pgfnodealias
1494       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1495       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496     }
1497     \endpgfpicture
1498   }
1499 }
```

```
1500 \cs_new_protected:Npn \@@_in_everycr:
1501 {
1502   \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1503   \tbl_update_cell_data_for_next_row:
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     \@@_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 \@@_renew_dots:
1528 {
1529     \cs_set_eq:NN \ldots \@@_Ldots:
1530     \cs_set_eq:NN \cdots \@@_Cdots:
1531     \cs_set_eq:NN \vdots \@@_Vdots:
1532     \cs_set_eq:NN \ddots \@@_Ddots:
1533     \cs_set_eq:NN \iddots \@@_Iddots:
1534     \cs_set_eq:NN \dots \@@_Ldots:
1535     \cs_set_eq:NN \hdotsfor \@@_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 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 {

```

<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}`.

```

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 }

```

`\@@_pre_array_after_CodeBefore:` will be executed in `\@@_pre_array:` *after* the execution of the `\CodeBefore`. It contains all the code before the beginning of the construction of `\l_@@_the_array_box`.

```

1556 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1557 {

```

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.

```

1558 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1559 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

1560 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1561 \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.

```

1562 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1563 \fp_gzero:N \g_@@_total_X_weight_fp
1564 \bool_gset_false:N \g_@@_V_of_X_bool

1565 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1566 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol

1567 \@@_patch_booktabs:
1568 \box_clear_new:N \l_@@_cell_box
1569 \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}`).

```

1570 \bool_if:NT \l_@@_small_bool
1571 {
1572     \def \arraystretch { 0.47 }
1573     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1574 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1575 }

```

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!

```

1576 \bool_if:NT \g_@@_create_cell_nodes_bool
1577 {
1578   \tl_put_right:Nn \@@_begin_of_row:
1579   {
1580     \pgfsys@markposition
1581     { \@@_env: - row - \int_use:N \c@iRow - base }
1582   }
1583   \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1584 }

```

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`.

```

1585 \bool_if:NF \c_@@_revtex_bool
1586 {
1587   \def \ar@ialign
1588   {
1589     \IfPackageLoadedT { latex-lab-testphase-table }
1590     { \tbl_init_cell_data_for_table: }
1591     \@@_some_initialization:
1592     \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`.

```

1593     \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1594     \halign
1595   }
1596 }

```

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).

```

1597 \bool_if:NT \c_@@_revtex_bool
1598 {
1599   \IfPackageLoadedT { colortbl }
1600   { \cs_set_protected:Npn \CT@setup { } }
1601 }

```

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).

```

1602 \cs_set_eq:NN \@@_old_ldots: \ldots
1603 \cs_set_eq:NN \@@_old_cdots: \cdots
1604 \cs_set_eq:NN \@@_old_vdots: \vdots
1605 \cs_set_eq:NN \@@_old_ddots: \ddots
1606 \cs_set_eq:NN \@@_old_iddots: \iddots
1607 \bool_if:NTF \l_@@_standard_cline_bool
1608 { \cs_set_eq:NN \cline \@@_standard_cline: }
1609 { \cs_set_eq:NN \cline \@@_cline: }
1610 \cs_set_eq:NN \Ldots \@@_Ldots:
1611 \cs_set_eq:NN \Cdots \@@_Cdots:
1612 \cs_set_eq:NN \Vdots \@@_Vdots:
1613 \cs_set_eq:NN \Ddots \@@_Ddots:
1614 \cs_set_eq:NN \Iddots \@@_Iddots:
1615 \cs_set_eq:NN \Hline \@@_Hline:
1616 \cs_set_eq:NN \Hspace \@@_Hspace:

```

```

1617 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1618 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1619 \cs_set_eq:NN \Block \@@_Block:
1620 \cs_set_eq:NN \rotate \@@_rotate:
1621 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1622 \cs_set_eq:NN \dotfill \@@_dotfill:
1623 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1624 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1625 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1626 \cs_set_eq:NN \TopRule \@@_TopRule
1627 \cs_set_eq:NN \MidRule \@@_MidRule
1628 \cs_set_eq:NN \BottomRule \@@_BottomRule
1629 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1630 \cs_set_eq:NN \Hbrace \@@_Hbrace
1631 \cs_set_eq:NN \Vbrace \@@_Vbrace
1632 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1633 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1634 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1635 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1636 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1637 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1638 \int_compare:nNt { \l_@@_first_row_int } > { \c_zero_int }
1639 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1640 \int_compare:nNt { \l_@@_last_row_int } < { \c_zero_int }
1641 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1642 \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:`.

```

1643 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1644 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1645 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1646 \@@_revert_colortbl:

```

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

```

1647 \tl_if_exist:NT \l_@@_note_in_caption_tl
1648 {
1649   \tl_if_empty:NF \l_@@_note_in_caption_tl
1650   {
1651     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1652     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1653   }
1654 }

```

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).

```

1655 \seq_gclear:N \g_@@_multicolumn_cells_seq
1656 \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).

```

1657 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1658 \int_gzero:N \g_@@_row_total_int

```



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

```

1659 \int_gzero:N \g_@@_col_total_int
1660 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1661 \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.

```

1662 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1663 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1664 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1665 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1666 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1667 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1668 \tl_gclear:N \g_nicematrix_code_before_tl
1669 \tl_gclear:N \g_@@_pre_code_before_tl

```

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

```

1670 \dim_zero_new:N \l_@@_left_delim_dim
1671 \dim_zero_new:N \l_@@_right_delim_dim
1672 \bool_if:NTF \g_@@_delims_bool
1673 {

```

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

```

1674 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1675 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1676 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1677 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1678 }
1679 {
1680 \dim_gset:Nn \l_@@_left_delim_dim
1681 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1682 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1683 }
1684 }

```

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

The command `\@@_pre_array:` will be executed after analysis of the keys of the environment. If will, in particular, read the potential informations written on the aux file.

```

1685 \cs_new_protected:Npn \@@_pre_array:
1686 {
1687 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1688 \int_gzero_new:N \c@iRow
1689 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1690 \int_gzero_new:N \c@jCol

```

We give values to the LaTeX counters `iRow` and `jCol`. We remind that before and after the main array (in particular in the `\CodeBefore` and the `\CodeAfter`, they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1691 \int_compare:nNnT \l_@@_last_row_int > 0
1692 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1693 \int_compare:nNnT \l_@@_last_col_int > 0
1694 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1695 \bool_if:NT \g_@@_aux_found_bool
1696 {
1697 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }

```

```

1698 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1699 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1700 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1701 }

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_col_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-col` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

1702 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1703 {
1704   \bool_set_true:N \l_@@_last_row_without_value_bool
1705   \bool_if:NT \g_@@_aux_found_bool
1706     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1707 }
1708 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1709 {
1710   \bool_if:NT \g_@@_aux_found_bool
1711     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1712 }

```

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”.

```

1713 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1714 {
1715   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1716     {
1717       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1718         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1719       \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1720         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1721     }
1722 }

1723 \seq_gclear:N \g_@@_cols_vlism_seq
1724 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1725 \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1726 \@@_pre_array_after_CodeBefore:

```

Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `\c_math_toggle_token` also).

```

1727 \hbox_set:Nw \l_@@_the_array_box
1728 \skip_horizontal:N \l_@@_left_margin_dim
1729 \skip_horizontal:N \l_@@_extra_left_margin_dim
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 {

```

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.

```

1748 \pgfsys@markposition { \@@_env: - position }
1749 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1750 \pgfpicture
1751 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1752 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1753 {
1754   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1755   \pgfcoordinate { \@@_env: - row - ##1 }
1756     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1757 }

```

Now, the recreation of the `col` nodes.

```

1758 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1759 {
1760   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1761   \pgfcoordinate { \@@_env: - col - ##1 }
1762     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1763 }

```

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

```

1764 \@@_create_diag_nodes:

```

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

```
1765 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1766 \endpgfpicture
```

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

```
1767 \@@_create_blocks_nodes:
1768 \IfPackageLoadedT { tikz }
1769 {
1770   \tikzset
1771   {
1772     every-picture / .style =
1773     { overlay , name-prefix = \@@_env: - }
1774   }
1775 }
1776 \cs_set_eq:NN \cellcolor \@@_cellcolor
1777 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1778 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1779 \cs_set_eq:NN \rowcolor \@@_rowcolor
1780 \cs_set_eq:NN \rowcolors \@@_rowcolors
1781 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1782 \cs_set_eq:NN \arraycolor \@@_arraycolor
1783 \cs_set_eq:NN \columncolor \@@_columncolor
1784 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1785 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1786 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1787 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1788 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1789 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1790 }
```

```
1791 \cs_new_protected:Npn \@@_exec_code_before:
1792 {
```

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...

```
1793 \clist_map_inline:Nn \l_@@_corners_cells_clist
1794 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1795 \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`.

```
1796 \@@_add_to_colors_seq:nn { { nocolor } } { }
1797 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1798 \group_begin:
```

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

```
1799 \if_mode_math:
1800 \@@_exec_code_before_i:
1801 \else:
1802 \c_math_toggle_token
1803 \@@_exec_code_before_i:
1804 \c_math_toggle_token
1805 \fi:
1806 \group_end:
1807 }
```

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`).

```

1808 \cs_new_protected:Npn \@@_exec_code_before_i:
1809 {
1810   \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1811   { \@@_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`.

```

1812   \exp_last_unbraced:No \@@_CodeBefore_keys:
1813   \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.

```

1814   \@@_actually_color:
1815   \l_@@_code_before_tl
1816   \q_stop
1817 }

```

```

1818 \keys_define:nn { nicematrix / CodeBefore }
1819 {
1820   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1821   create-cell-nodes .default:n = true ,
1822   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1823   sub-matrix .value_required:n = true ,
1824   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1825   delimiters / color .value_required:n = true ,
1826   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1827 }
1828 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1829 {
1830   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1831   \@@_CodeBefore:w
1832 }

```

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.

```

1833 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1834 {
1835   \bool_if:NT \g_@@_aux_found_bool
1836   {
1837     \@@_pre_code_before:
1838     \legacy_if:nF { measuring@ } { #1 }
1839   }
1840 }

```

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.

```

1841 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1842 {
1843   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1844   {
1845     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:

```

```

1846 \pgfcoordinate { \@@_env: - row - ##1 - base }
1847 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1848 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1849 {
1850 \cs_if_exist:cT
1851 { \pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1852 {
1853 \pgfsys@getposition
1854 { \@@_env: - ##1 - #####1 - NW }
1855 \@@_node_position:
1856 \pgfsys@getposition
1857 { \@@_env: - ##1 - #####1 - SE }
1858 \@@_node_position_i:
1859 \@@_pgf_rect_node:nnn
1860 { \@@_env: - ##1 - #####1 }
1861 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1862 { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1863 }
1864 }
1865 }
1866 \@@_create_extra_nodes:
1867 \@@_create_aliases_last:
1868 }

1869 \cs_new_protected:Npn \@@_create_aliases_last:
1870 {
1871 \int_step_inline:nn { \c@iRow }
1872 {
1873 \pgfnodealias
1874 { \@@_env: - ##1 - last }
1875 { \@@_env: - ##1 - \int_use:N \c@jCol }
1876 }
1877 \int_step_inline:nn { \c@jCol }
1878 {
1879 \pgfnodealias
1880 { \@@_env: - last - ##1 }
1881 { \@@_env: - \int_use:N \c@iRow - ##1 }
1882 }
1883 \pgfnodealias
1884 { \@@_env: - last - last }
1885 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1886 }

1887 \cs_new_protected:Npn \@@_create_blocks_nodes:
1888 {
1889 \pgfpicture
1890 \pgf@relevantforpicturesizefalse
1891 \pgfrememberpicturepositiononpagetrue
1892 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1893 { \@@_create_one_block_node:nnnnn ##1 }
1894 \endpgfpicture
1895 }

```

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>

```

1896 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1897 {
1898 \tl_if_empty:nF { #5 }

```

---

<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).

```

1899 {
1900   \@@_qpoint:n { col - #2 }
1901   \dim_set_eq:NN \l_tmpa_dim \pgf@x
1902   \@@_qpoint:n { #1 }
1903   \dim_set_eq:NN \l_tmpb_dim \pgf@y
1904   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1905   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1906   \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1907   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1908   \@@_pgf_rect_node:nnnnn
1909     { \@@_env: - #5 }
1910     { \dim_use:N \l_tmpa_dim }
1911     { \dim_use:N \l_tmpb_dim }
1912     { \dim_use:N \l_@@_tmpc_dim }
1913     { \dim_use:N \l_@@_tmpd_dim }
1914   }
1915 }

1916 \cs_new_protected:Npn \@@_patch_for_revtext:
1917 {
1918   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1919   \cs_set_eq:NN \@array \@array@array
1920   \cs_set_eq:NN \@tabular \@tabular@array
1921   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1922   \cs_set_eq:NN \array \array@array
1923   \cs_set_eq:NN \endarray \endarray@array
1924   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1925   \cs_set_eq:NN \@mkpream \@mkpream@array
1926   \cs_set_eq:NN \@classx \@classx@array
1927   \cs_set_eq:NN \insert@column \insert@column@array
1928   \cs_set_eq:NN \@arraycr \@arraycr@array
1929   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1930   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1931 }

```

## 10 The environment {NiceArrayWithDelims}

```

1932 \NewDocumentEnvironment { NiceArrayWithDelims }
1933 { m m 0 { } m ! 0 { } t \CodeBefore }
1934 {
1935   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
1936   \@@_provide_pgfsyspdfmark:
1937   \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.

```

1938   \bgroup

1939   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1940   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1941   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1942   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1943   \int_gzero:N \g_@@_block_box_int
1944   \dim_gzero:N \g_@@_width_last_col_dim
1945   \dim_gzero:N \g_@@_width_first_col_dim
1946   \bool_gset_false:N \g_@@_row_of_col_done_bool
1947   \str_if_empty:NT \g_@@_name_env_str

```

```

1948     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1949     \bool_if:NTF \l_@@_tabular_bool
1950     { \mode_leave_vertical: }
1951     { \@@_test_if_math_mode: }
1952     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1953     \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.

```

1954     \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.

```

1955     \cs_if_exist:NT \tikz@library@external@loaded
1956     {
1957         \tikzexternaldisable
1958         \cs_if_exist:NT \ifstandalone
1959         { \tikzset { external / optimize = false } }
1960     }

```

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

```

1961     \int_gincr:N \g_@@_env_int
1962     \bool_if:NF \l_@@_block_auto_columns_width_bool
1963     { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

The sequence `\g_@@_blocks_seq` will contain the characteristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks.

```

1964     \seq_gclear:N \g_@@_blocks_seq
1965     \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`.

```

1966     \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1967     \seq_gclear:N \g_@@_pos_of_xdots_seq
1968     \tl_gclear_new:N \g_@@_code_before_tl
1969     \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.

```

1970     \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1971     {
1972         \bool_gset_true:N \g_@@_aux_found_bool
1973         \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1974     }
1975     { \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.

```

1976     \tl_gclear:N \g_@@_aux_tl
1977     \tl_if_empty:NF \g_@@_code_before_tl
1978     {
1979         \bool_set_true:N \l_@@_code_before_bool
1980         \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1981     }
1982     \tl_if_empty:NF \g_@@_pre_code_before_tl
1983     { \bool_set_true:N \l_@@_code_before_bool }

```

---

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



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`.

```

1984   \bool_if:NTF \g_@@_delims_bool
1985   { \keys_set:nn { nicematrix / pNiceArray } }
1986   { \keys_set:nn { nicematrix / NiceArray } }
1987   { #3 , #5 }

1988   \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302

```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It’s the job that will do the command `\@@_CodeBefore_Body:w`. After that job, the command `\@@_CodeBefore_Body:w` will go on with `\@@_pre_array:`.

```

1989   \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
1990 }

```

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

```

1991 {
1992   \bool_if:NTF \l_@@_light_syntax_bool
1993   { \use:c { end @@-light-syntax } }
1994   { \use:c { end @@-normal-syntax } }
1995   \c_math_toggle_token
1996   \skip_horizontal:N \l_@@_right_margin_dim
1997   \skip_horizontal:N \l_@@_extra_right_margin_dim
1998   \hbox_set_end:
1999   \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.

```

2000   \bool_if:NT \l_@@_width_used_bool
2001   {
2002     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2003     { \@@_error_or_warning:n { width-without-X-columns } }
2004   }

```

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  $x$ .

```

2005   \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2006   { \@@_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).

```

2007   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2008   {
2009     \bool_if:NF \l_@@_last_row_without_value_bool
2010     {
2011       \int_compare:nNnF { \l_@@_last_row_int } = { \c_iRow }
2012       {
2013         \@@_error:n { Wrong-last-row }
2014         \int_set_eq:NN \l_@@_last_row_int \c_iRow
2015       }
2016     }
2017   }

```

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

```

2018 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2019 \bool_if:NTF \g_@@_last_col_found_bool
2020 { \int_gdecr:N \c@jCol }
2021 {
2022 \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2023 { \@@_error:n { last~col~not~used } }
2024 }

```

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

```

2025 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2026 \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2027 { \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. 92).

```

2028 \int_if_zero:nT { \l_@@_first_col_int }
2029 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2030 \bool_if:nTF { ! \g_@@_delims_bool }
2031 {
2032 \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2033 { \@@_use_arraybox_with_notes_c: }
2034 {
2035 \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2036 { \@@_use_arraybox_with_notes_b: }
2037 { \@@_use_arraybox_with_notes: }
2038 }
2039 }

```

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).

```

2040 {
2041 \int_if_zero:nTF { \l_@@_first_row_int }
2042 {
2043 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2044 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2045 }
2046 { \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>

```

2047 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2048 {
2049 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2050 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2051 }
2052 { \dim_zero:N \l_tmpb_dim }
2053 \hbox_set:Nn \l_tmpa_box
2054 {
2055 \m@th
2056 \c_math_toggle_token
2057 \@@_color:o \l_@@_delimiters_color_tl
2058 \exp_after:wN \left \g_@@_left_delim_tl
2059 \vcenter
2060 {

```

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

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

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```

2061         \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2062         \hbox
2063         {
2064             \bool_if:NTF \l_@@_tabular_bool
2065             { \skip_horizontal:n { - \tabcolsep } }
2066             { \skip_horizontal:n { - \arraycolsep } }
2067             \@@_use_arraybox_with_notes_c:
2068             \bool_if:NTF \l_@@_tabular_bool
2069             { \skip_horizontal:n { - \tabcolsep } }
2070             { \skip_horizontal:n { - \arraycolsep } }
2071         }

```

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

```

2072         \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2073     }
2074     \exp_after:wN \right \g_@@_right_delim_tl
2075     \c_math_toggle_token
2076 }

```

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.

```

2077     \bool_if:NTF \l_@@_delimiters_max_width_bool
2078     {
2079         \@@_put_box_in_flow_bis:nn
2080         { \g_@@_left_delim_tl }
2081         { \g_@@_right_delim_tl }
2082     }
2083     \@@_put_box_in_flow:
2084 }

```

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. 93).

```

2085     \bool_if:NT \g_@@_last_col_found_bool
2086     { \skip_horizontal:N \g_@@_width_last_col_dim }
2087     \bool_if:NT \l_@@_preamble_bool
2088     {
2089         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2090         { \@@_err_columns_not_used: }
2091     }
2092     \@@_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.

```

2093     \egroup

```

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

```

2094     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2095     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2096     \iow_now:Ne \@mainaux
2097     {
2098         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2099         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2100         { \exp_not:o \g_@@_aux_tl }
2101     }
2102     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2103     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2104 }

```

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

```

2105 \cs_new_protected:Npn \@@_err_columns_not_used:

```

```

2106 {
2107   \@@_warning:n { columns-not-used }
2108   \cs_gset:Npn \@@_err_columns_not_used: { }
2109 }

```

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  $x$ .

```

2110 \cs_new_protected:Npn \@@_compute_width_X:
2111 {
2112   \tl_gput_right:Ne \g_@@_aux_tl
2113   {
2114     \bool_set_true:N \l_@@_X_columns_aux_bool
2115     \dim_set:Nn \l_@@_X_columns_dim
2116     {

```

The flag `\g_@@_V_of_X_bool` is raised when there is at least in the tabular a column of type X using the key V. In that case, the width of the X column may be considered as correct even though the tabular has not (of course) a width equal to `\l_@@_width_dim`

```

2117       \bool_lazy_and:nnTF
2118       { \g_@@_V_of_X_bool }
2119       { \l_@@_X_columns_aux_bool }
2120       { \dim_use:N \l_@@_X_columns_dim }
2121       {
2122         \dim_compare:nNnTF
2123         {
2124           \dim_abs:n
2125           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2126         }
2127         <
2128         { 0.001 pt }
2129         { \dim_use:N \l_@@_X_columns_dim }
2130         {
2131           \dim_eval:n
2132           {
2133             \l_@@_X_columns_dim
2134             +
2135             \fp_to_dim:n
2136             {
2137               (
2138                 \dim_eval:n
2139                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2140               )
2141               / \fp_use:N \g_@@_total_X_weight_fp
2142             }
2143           }
2144         }
2145       }
2146     }
2147   }
2148 }

```

## 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`.

```

2149 \cs_new_protected:Npn \@@_transform_preamble:
2150 {
2151   \@@_transform_preamble_i:
2152   \@@_transform_preamble_ii:
2153 }
2154 \cs_new_protected:Npn \@@_transform_preamble_i:
2155 {
2156   \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`).

```

2157   \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.

```

2158   \bool_gset_false:N \g_tmpb_bool

```

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

```

2159   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2160   \int_zero:N \l_tmpa_int
2161   \tl_gclear:N \g_@@_array_preamble_tl
2162   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2163   {
2164     \tl_gset:Nn \g_@@_array_preamble_tl
2165     { ! { \skip_horizontal:N \arrayrulewidth } }
2166   }
2167   {
2168     \clist_if_in:NnT \l_@@_vlines_clist 1
2169     {
2170       \tl_gset:Nn \g_@@_array_preamble_tl
2171       { ! { \skip_horizontal:N \arrayrulewidth } }
2172     }
2173   }

```

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

```

2174   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2175   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2176   \@@_replace_columncolor:
2177 }

```

```

2178 \cs_new_protected:Npn \@@_transform_preamble_ii:
2179 {

```

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

```

2180   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2181   {
2182     \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2183     { \bool_gset_true:N \g_@@_delims_bool }
2184   }
2185   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2186   \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).

```

2187 \int_if_zero:nTF { \l_@@_first_col_int }
2188 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2189 {
2190   \bool_if:NF \g_@@_delims_bool
2191   {
2192     \bool_if:NF \l_@@_tabular_bool
2193     {
2194       \clist_if_empty:NT \l_@@_vlines_clist
2195       {
2196         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2197         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2198       }
2199     }
2200   }
2201 }
2202 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2203 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2204 {
2205   \bool_if:NF \g_@@_delims_bool
2206   {
2207     \bool_if:NF \l_@@_tabular_bool
2208     {
2209       \clist_if_empty:NT \l_@@_vlines_clist
2210       {
2211         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2212         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2213       }
2214     }
2215   }
2216 }

```

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`).

```

2217 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2218 {

```

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

```

2219 \IfPackageLoadedF { latex-lab-testphase-table }
2220 {
2221   \tl_gput_right:Nn \g_@@_array_preamble_tl
2222   { > { \@@_error_too_much_cols: } l }
2223 }
2224 }
2225 }

```

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.

```

2226 \cs_new_protected:Npn \@@_rec_preamble:n #1
2227 {

```

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>

```

2228 \cs_if_exist:cTF { @@_ \token_to_str:N #1 : }
2229 { \use:c { @@_ \token_to_str:N #1 : } { #1 } }
2230 {

```

---

<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`.

Now, the columns defined by \newcolumnntype of array.

```

2231     \cs_if_exist:cTF { NC @ find @ #1 }
2232     {
2233         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2234         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2235     }
2236     {
2237         \str_if_eq:nnTF { #1 } { S }
2238         { \@@_fatal:n { unknown~column~type~S } }
2239         { \@@_fatal:nn { unknown~column~type } { #1 } }
2240     }
2241 }
2242 }

```

For c, l and r

```

2243 \cs_new_protected:Npn \@@_c: #1
2244 {
2245     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2246     \tl_gclear:N \g_@@_pre_cell_tl
2247     \tl_gput_right:Nn \g_@@_array_preamble_tl
2248     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2249     \int_gincr:N \c@jCol
2250     \@@_rec_preamble_after_col:n
2251 }

2252 \cs_new_protected:Npn \@@_l: #1
2253 {
2254     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2255     \tl_gclear:N \g_@@_pre_cell_tl
2256     \tl_gput_right:Nn \g_@@_array_preamble_tl
2257     {
2258         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2259         l
2260         < \@@_cell_end:
2261     }
2262     \int_gincr:N \c@jCol
2263     \@@_rec_preamble_after_col:n
2264 }

2265 \cs_new_protected:Npn \@@_r: #1
2266 {
2267     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2268     \tl_gclear:N \g_@@_pre_cell_tl
2269     \tl_gput_right:Nn \g_@@_array_preamble_tl
2270     {
2271         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2272         r
2273         < \@@_cell_end:
2274     }
2275     \int_gincr:N \c@jCol
2276     \@@_rec_preamble_after_col:n
2277 }

```

For ! and @

```

2278 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2279 {
2280     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2281     \@@_rec_preamble:n
2282 }
2283 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2284 \cs_new_protected:cpn { @@ _ | : } #1
2285 {
\l_tmpa_int is the number of successive occurrences of |
2286 \int_incr:N \l_tmpa_int
2287 \@@_make_preamble_i_i:n
2288 }
2289 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2290 {

```

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

```

2291 \str_if_eq:nnTF { #1 } { | }
2292 { \use:c { @@ _ | : } | }
2293 { \@@_make_preamble_i_ii:nn { } #1 }
2294 }
2295 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2296 {
2297 \str_if_eq:nnTF { #2 } { [ ]
2298 { \@@_make_preamble_i_ii:nw { #1 } [ ]
2299 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2300 }
2301 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2302 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2303 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2304 {
2305 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2306 \tl_gput_right:Ne \g_@@_array_preamble_tl
2307 {

```

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

```

2308 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2309 }
2310 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2311 {
2312 \@@_vline:n
2313 {
2314 position = \int_eval:n { \c@jCol + 1 } ,
2315 multiplicity = \int_use:N \l_tmpa_int ,
2316 total-width = \dim_use:N \l_@@_rule_width_dim ,
2317 #2
2318 }

```

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.

```

2319 }
2320 \int_zero:N \l_tmpa_int
2321 \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2322 \@@_rec_preamble:n #1
2323 }
2324 \cs_new_protected:cpn { @@ _ > : } #1 #2
2325 {
2326 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2327 \@@_rec_preamble:n
2328 }
2329 \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.

```

2330 \keys_define:nn { nicematrix / p-column }
2331 {
2332   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2333   r .value_forbidden:n = true ,
2334   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2335   c .value_forbidden:n = true ,
2336   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2337   l .value_forbidden:n = true ,
2338   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2339   S .value_forbidden:n = true ,
2340   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2341   p .value_forbidden:n = true ,
2342   t .meta:n = p ,
2343   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2344   m .value_forbidden:n = true ,
2345   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2346   b .value_forbidden:n = true
2347 }

```

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

```

2348 \cs_new_protected:Npn \@@_p: #1
2349 {
2350   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2351   \@@_make_preamble_ii_i:n
2352 }
2353 \cs_set_eq:NN \@@_b: \@@_p:
2354 \cs_set_eq:NN \@@_m: \@@_p:
2355 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2356 {
2357   \str_if_eq:nnTF { #1 } { [ ]
2358     { \@@_make_preamble_ii_ii:w [ ]
2359       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2360     }
2361   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2362   { \@@_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.

```

2363 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2364 {

```

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).

```

2365   \str_set:Nn \l_@@_hpos_col_str { j }
2366   \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2367   \setlength { \l_tmpa_dim } { #2 }
2368   \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2369 }
2370 \cs_new_protected:Npn \@@_keys_p_column:n #1
2371 { \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.

```
2372 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2373 {
```

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

```
2374 \use:e
2375 {
2376 \@@_make_preamble_ii_vi:nnnnnnnn
2377 { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2378 { #1 }
2379 {
```

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.

```
2380 \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2381 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2382 {
```

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

```
2383 \def \exp_not:N \l_@@_hpos_cell_tl
2384 { \str_lowercase:f { \l_@@_hpos_col_str } }
2385 }
2386 \IfPackageLoadedTF { ragged2e }
2387 {
2388 \str_case:on \l_@@_hpos_col_str
2389 {
```

The following `\exp_not:N` are mandatory.

```
2390 c { \exp_not:N \Centering }
2391 l { \exp_not:N \RaggedRight }
2392 r { \exp_not:N \RaggedLeft }
2393 }
2394 }
2395 {
2396 \str_case:on \l_@@_hpos_col_str
2397 {
2398 c { \exp_not:N \centering }
2399 l { \exp_not:N \raggedright }
2400 r { \exp_not:N \raggedleft }
2401 }
2402 }
2403 #3
2404 }
2405 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2406 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2407 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2408 { #2 }
2409 {
2410 \str_case:onF \l_@@_hpos_col_str
2411 {
2412 { j } { c }
2413 { si } { c }
2414 }
```

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

```
2415 { \str_lowercase:f \l_@@_hpos_col_str }
2416 }
2417 }
```

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

```
2418 \int_gincr:N \c@jCol
2419 \@@_rec_preamble_after_col:n
2420 }
```

#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*.

```

2421 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2422 {
2423   \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2424   {
2425     \tl_gput_right:Nn \g_@@_array_preamble_tl
2426     { > \@@_test_if_empty_for_S: }
2427   }
2428   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2429   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2430   \tl_gclear:N \g_@@_pre_cell_tl
2431   \tl_gput_right:Nn \g_@@_array_preamble_tl
2432   {
2433     > {

```

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.

```

2434   \dim_set:Nn \l_@@_col_width_dim { #2 }
2435   \IfPackageLoadedT { latex-lab-testphase-table }
2436   { \tag_struct_begin:n { tag = Div } }
2437   \@@_cell_begin:

```

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

```

2438   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2439   \everypar
2440   {
2441     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2442     \everypar { }
2443   }
2444   \IfPackageLoadedT { latex-lab-testphase-table }
2445   { \tagpdfpara0n }

```

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

```

2446   #3

```

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

```

2447   \g_@@_row_style_tl
2448   \arraybackslash
2449   #5
2450   }
2451   #8
2452   < {
2453   #6

```

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

```

2454   \@finalstrut \@arstrutbox
2455   \use:c { end #7 }

```

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

```

2456         #4
2457         \@@_cell_end:
2458         \IfPackageLoadedT { latex-lab-testphase-table }
2459         { \tag_struct_end: }
2460     }
2461 }
2462 }

```

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

```

2463 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2464 {

```

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 &).

```

2465     \group_align_safe_begin:
2466     \peek_meaning:NTF &
2467     { \@@_the_cell_is_empty: }
2468     {
2469         \peek_meaning:NTF \\\
2470         { \@@_the_cell_is_empty: }
2471         {
2472             \peek_meaning:NTF \crcr
2473             \@@_the_cell_is_empty:
2474             \group_align_safe_end:
2475         }
2476     }
2477 }
2478 \cs_new_protected:Npn \@@_the_cell_is_empty:
2479 {
2480     \group_align_safe_end:
2481     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2482     {

```

Be careful: here, we can't merely use \bool\_gset\_true: \g\_@@\_empty\_cell\_bool, in particular because of the columns of type X.

```

2483         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2484         \skip_horizontal:N \l_@@_col_width_dim
2485     }
2486 }
2487 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2488 {
2489     \peek_meaning:NT \__siunitx_table_skip:n
2490     { \bool_gset_true:N \g_@@_empty_cell_bool }
2491 }

```

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.

```

2492 \cs_new_protected:Npn \@@_center_cell_box:
2493 {

```

By putting instructions in \g\_@@\_cell\_after\_hook\_tl, we require a post-action of the box \l\_@@\_cell\_box.

```

2494     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2495     {
2496         \dim_compare:nNnT
2497         { \box_ht:N \l_@@_cell_box }
2498         >

```

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).

```

2499     { \box_ht:N \strutbox }
2500     {
2501       \hbox_set:Nn \l_@@_cell_box
2502       {
2503         \box_move_down:nn
2504         {
2505           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2506             + \baselineskip ) / 2
2507         }
2508         { \box_use:N \l_@@_cell_box }
2509       }
2510     }
2511   }
2512 }

```

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

```

2513 \cs_new_protected:Npn \@_V: #1 #2
2514 {
2515   \str_if_eq:nnTF { #2 } { [ ] }
2516   { \@@_make_preamble_V_i:w [ ] }
2517   { \@@_make_preamble_V_i:w [ ] { #2 } }
2518 }
2519 \cs_new_protected:Npn \@_make_preamble_V_i:w [ #1 ]
2520 { \@@_make_preamble_V_ii:nn { #1 } }
2521 \cs_new_protected:Npn \@_make_preamble_V_ii:nn #1 #2
2522 {
2523   \str_set:Nn \l_@@_vpos_col_str { p }
2524   \str_set:Nn \l_@@_hpos_col_str { j }
2525   \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2526   \setlength { \l_tmpa_dim } { #2 }
2527   \IfPackageLoadedTF { varwidth }
2528   { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2529   {
2530     \@@_error_or_warning:n { varwidth-not-loaded }
2531     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2532   }
2533 }

```

For `w` and `W`

```

2534 \cs_new_protected:Npn \@_w: { \@@_make_preamble_w:nnnn { } }
2535 \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.

```

2536 \cs_new_protected:Npn \@_make_preamble_w:nnnn #1 #2 #3 #4
2537 {
2538   \str_if_eq:nnTF { #3 } { s }
2539   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2540   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2541 }

```

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.

```

2542 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2543 {
2544   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2545   \tl_gclear:N \g_@@_pre_cell_tl
2546   \tl_gput_right:Nn \g_@@_array_preamble_tl
2547   {
2548     > {

```

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

```

2549       \setlength { \l_@@_col_width_dim } { #2 }
2550       \@@_cell_begin:
2551       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2552     }
2553     c
2554     < {
2555       \@@_cell_end_for_w_s:
2556       #1
2557       \@@_adjust_size_box:
2558       \box_use_drop:N \l_@@_cell_box
2559     }
2560   }
2561   \int_gincr:N \c@jCol
2562   \@@_rec_preamble_after_col:n
2563 }

```

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

```

2564 \cs_new_protected:Npn \@@_make_preamble_w_iii:nnnn #1 #2 #3 #4
2565 {
2566   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2567   \tl_gclear:N \g_@@_pre_cell_tl
2568   \tl_gput_right:Nn \g_@@_array_preamble_tl
2569   {
2570     > {

```

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

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

```

2571       \setlength { \l_@@_col_width_dim } { #4 }
2572       \hbox_set:Nw \l_@@_cell_box
2573       \@@_cell_begin:
2574       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2575     }
2576     c
2577     < {
2578       \@@_cell_end:
2579       \hbox_set_end:
2580       #1
2581       \@@_adjust_size_box:
2582       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2583     }
2584   }

```

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

```

2585   \int_gincr:N \c@jCol
2586   \@@_rec_preamble_after_col:n
2587 }

```

```

2588 \cs_new_protected:Npn \@@_special_W:
2589 {
2590   \dim_compare:nNt { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2591   { \@@_warning:n { W~warning } }
2592 }

```

For S (of siunitx).

```

2593 \cs_new_protected:Npn \@@_S: #1 #2
2594 {
2595   \str_if_eq:nnTF { #2 } { [ ] }
2596   { \@@_make_preamble_S:w [ ] }
2597   { \@@_make_preamble_S:w [ ] { #2 } }
2598 }
2599 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2600 { \@@_make_preamble_S_i:n { #1 } }
2601 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2602 {
2603   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2604   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2605   \tl_gclear:N \g_@@_pre_cell_tl
2606   \tl_gput_right:Nn \g_@@_array_preamble_tl
2607   {
2608     > {

```

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).

```

2609       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2610       \keys_set:nn { siunitx } { #1 }
2611       \@@_cell_begin:
2612       \siunitx_cell_begin:w
2613     }
2614     c
2615     <
2616     {
2617       \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`).

```

2618       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2619       {
2620         \bool_if:NTF \l__siunitx_table_text_bool
2621         { \bool_set_true:N }
2622         { \bool_set_false:N }
2623         \l__siunitx_table_text_bool
2624       }
2625       \@@_cell_end:
2626     }
2627   }

```

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

```

2628     \int_gincr:N \c@jCol
2629     \@@_rec_preamble_after_col:n
2630   }

```

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

```

2631 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2632 {
2633   \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.

```

2634 \int_if_zero:nTF { \c@jCol }
2635 {
2636   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2637   {

```

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

```

2638     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2639     \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2640     \@@_rec_preamble:n #2
2641   }
2642   {
2643     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2644     \@@_make_preamble_iv:nn { #1 } { #2 }
2645   }
2646 }
2647 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2648 }
2649 \cs_set_eq:cc { @@ _ \token_to_str:N [ : ] { @@ _ \token_to_str:N ( : }
2650 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2651 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2652 {
2653   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2654   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2655   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2656   {
2657     \@@_error:nn { delimiter~after~opening } { #2 }
2658     \@@_rec_preamble:n
2659   }
2660   { \@@_rec_preamble:n #2 }
2661 }

```

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

```

2662 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2663 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2664 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2665 {
2666   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2667   \tl_if_in:nnTF { ) ] \} } { #2 }
2668   { \@@_make_preamble_v:nnn #1 #2 }
2669   {
2670     \str_if_eq:nnTF { \s_stop } { #2 }
2671     {
2672       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2673       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2674       {
2675         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2676         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2677         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2678         \@@_rec_preamble:n #2
2679       }
2680     }
2681     {
2682       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2683       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2684       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2685       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }

```



```

2686         \@@_rec_preamble:n #2
2687     }
2688 }
2689 }
2690 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2691 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2692 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2693 {
2694     \str_if_eq:nnTF { \s_stop } { #3 }
2695     {
2696         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2697         {
2698             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2699             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2700             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2701             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2702         }
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             \@@_error:nn { double~closing~delimiter } { #2 }
2708         }
2709     }
2710     {
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         \@@_rec_preamble:n #3
2715     }
2716 }
2717 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2718 { \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 @{...}.

```

2719 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2720 {
2721     \str_if_eq:nnTF { #1 } { < }
2722     { \@@_rec_preamble_after_col_i:n }
2723     {
2724         \str_if_eq:nnTF { #1 } { @ }
2725         { \@@_rec_preamble_after_col_ii:n }
2726         {
2727             \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2728             {
2729                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2730                 { ! { \skip_horizontal:N \arrayrulewidth } }
2731             }
2732             {
2733                 \clist_if_in:NeT \l_@@_vlines_clist
2734                 { \int_eval:n { \c@jCol + 1 } }
2735                 {
2736                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2737                     { ! { \skip_horizontal:N \arrayrulewidth } }
2738                 }
2739             }
2740             \@@_rec_preamble:n { #1 }
2741         }
2742     }
2743 }

```

```

2744 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2745 {
2746   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2747   \@@_rec_preamble_after_col:n
2748 }

```

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.

```

2749 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2750 {
2751   \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2752   {
2753     \tl_gput_right:Nn \g_@@_array_preamble_tl
2754     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2755   }
2756   {
2757     \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2758     {
2759       \tl_gput_right:Nn \g_@@_array_preamble_tl
2760       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2761     }
2762     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2763   }
2764   \@@_rec_preamble:n
2765 }

```

```

2766 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2767 {
2768   \tl_clear:N \l_tmpa_tl
2769   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2770   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2771 }

```

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.

```

2772 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2773 { \@@_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.

```

2774 \cs_new_protected:Npn \@@_X: #1 #2
2775 {
2776   \str_if_eq:nnTF { #2 } { [ ]
2777     { \@@_make_preamble_X:w [ ] }
2778     { \@@_make_preamble_X:w [ ] #2 }
2779   }
2780   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2781   { \@@_make_preamble_X_i:n { #1 } }

```

`#1` is the optional argument of the X specifier (a list of *key-value* pairs).

The following set of keys is for the specifier X in the preamble of the array. Such specifier may have as keys all the keys of `{ nicematrix / p-column }` but also a key V and also a key which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```

2782 \keys_define:nn { nicematrix / X-column }
2783 {
2784   V .code:n =
2785     \IfPackageLoadedTF { varwidth }
2786     {
2787       \bool_set_true:N \l_@@_V_of_X_bool

```

```

2788     \bool_gset_true:N \g_@@_V_of_X_bool
2789   }
2790   { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2791   unknown .code:n =
2792     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2793     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2794     { \@@_error_or_warning:n { invalid~weight } }
2795   }

```

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

```

2796 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2797 {

```

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).

```

2798   \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).

```

2799   \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`).

```

2800   \fp_set:Nn \l_tmpa_fp { 1.0 }
2801   \@@_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.

```

2802   \bool_set_false:N \l_@@_V_of_X_bool
2803   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2804   \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).

```

2805   \bool_if:NTF \l_@@_X_columns_aux_bool
2806   {
2807     \@@_make_preamble_ii_iv:nnn

```

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

```

2808     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2809     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2810     { \@@_no_update_width: }
2811   }

```

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.

```

2812   {
2813     \tl_gput_right:Nn \g_@@_array_preamble_tl
2814     {
2815       > {
2816         \@@_cell_begin:
2817         \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`.

```

2818     \NotEmpty

```

The following code will nullify the box of the cell.

```

2819     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2820     { \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`.

```

2821         \begin { minipage } { 5 cm } \arraybackslash
2822     }
2823     c
2824     < {
2825         \end { minipage }
2826         \@@_cell_end:
2827     }
2828 }
2829 \int_gincr:N \c@jCol
2830 \@@_rec_preamble_after_col:n
2831 }
2832 }

2833 \cs_new_protected:Npn \@@_no_update_width:
2834 {
2835     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2836     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2837 }

```

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

```

2838 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2839 {
2840     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2841     { \int_eval:n { \c@jCol + 1 } }
2842     \tl_gput_right:Ne \g_@@_array_preamble_tl
2843     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2844     \@@_rec_preamble:n
2845 }

```

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.

```

2846 \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).

```

2847 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2848 { \@@_fatal:n { Preamble-forgotten } }
2849 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2851 { @@ _ \token_to_str:N \hline : }
2852 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2853 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2854 { @@ _ \token_to_str:N \hline : }
2855 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2856 { @@ _ \token_to_str:N \hline : }
2857 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2858 { @@ _ \token_to_str:N \hline : }
2859 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2860 { @@ _ \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).

```

2861 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2862 {

```

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 that one column specifier in the preamble of `\multicolumn`.

```

2863 \multispan { #1 }
2864 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2865 \begingroup
2866 \IfPackageLoadedTF { latex-lab-testphase-table }
2867   { \tbl_update_multicolumn_cell_data:n { #1 } }
2868 \def \@@addamp
2869   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2870 \tl_gclear:N \g_@@_preamble_tl
2871 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2872 \exp_args:No \@mkpream \g_@@_preamble_tl
2873 \@addtopreamble \@empty
2874 \endgroup
2875 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

2876 \int_compare:nNnT { #1 } > { \c_one_int }
2877 {
2878   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2879   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2880   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2881   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2882   {
2883     {
2884       \int_if_zero:nTF { \c@jCol }
2885         { \int_eval:n { \c@iRow + 1 } }
2886         { \int_use:N \c@iRow }
2887     }
2888     { \int_eval:n { \c@jCol + 1 } }
2889     {
2890       \int_if_zero:nTF { \c@jCol }
2891         { \int_eval:n { \c@iRow + 1 } }
2892         { \int_use:N \c@iRow }
2893     }
2894     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2895   { }
2896 }
2897 }

```

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

```

2898 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2899 {
2900   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2901   {
2902     \@@_rectanglecolor [ ##1 ]
2903     { \exp_not:n { ##2 } }
2904     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2905     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2906   }
2907   \ignorespaces
2908 }

```

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

```
2909 \def \@sharp { #3 }
2910 \@arstrut
2911 \@preamble
2912 \null
```

We add some lines.

```
2913 \int_gadd:Nn \c@jCol { #1 - 1 }
2914 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
2915 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2916 \ignorespaces
2917 }
```

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`.

```
2918 \cs_new_protected:Npn \@_make_m_preamble:n #1
2919 {
2920   \str_case:nnF { #1 }
2921   {
2922     c { \@_make_m_preamble_i:n #1 }
2923     l { \@_make_m_preamble_i:n #1 }
2924     r { \@_make_m_preamble_i:n #1 }
2925     > { \@_make_m_preamble_ii:nn #1 }
2926     ! { \@_make_m_preamble_ii:nn #1 }
2927     @ { \@_make_m_preamble_ii:nn #1 }
2928     | { \@_make_m_preamble_iii:n #1 }
2929     p { \@_make_m_preamble_iv:nnn t #1 }
2930     m { \@_make_m_preamble_iv:nnn c #1 }
2931     b { \@_make_m_preamble_iv:nnn b #1 }
2932     w { \@_make_m_preamble_v:nnnn { } #1 }
2933     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2934     \q_stop { }
2935   }
2936   {
2937     \cs_if_exist:cTF { NC @ find @ #1 }
2938     {
2939       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2940       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2941     }
2942     {
2943       \str_if_eq:nnTF { #1 } { S }
2944       { \@_fatal:n { unknown~column~type~S~multicolumn } }
2945       { \@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2946     }
2947   }
2948 }
```

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

```
2949 \cs_new_protected:Npn \@_make_m_preamble_i:n #1
2950 {
2951   \tl_gput_right:Nn \g_@@_preamble_tl
2952   {
2953     > { \@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2954     #1
2955     < \@_cell_end:
2956   }
```

We test for the presence of a `<`.

```
2957 \@_make_m_preamble_x:n
2958 }
```

For >, ! and @

```

2959 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2960 {
2961   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2962   \@@_make_m_preamble:n
2963 }

```

For |

```

2964 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2965 {
2966   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2967   \@@_make_m_preamble:n
2968 }

```

For p, m and b

```

2969 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2970 {
2971   \tl_gput_right:Nn \g_@@_preamble_tl
2972   {
2973     > {
2974       \@@_cell_begin:

```

We use `\setlength` instead of `\dim_set:N` to allow a specifier like `p{\widthof{Some words}}`. `widthof` is a command provided by `calc`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2975       \setlength { \l_tmpa_dim } { #3 }
2976       \begin { minipage } [ #1 ] { \l_tmpa_dim }
2977       \mode_leave_vertical:
2978       \arraybackslash
2979       \vrule height \box_ht:N \@@arstrutbox depth \c_zero_dim width \c_zero_dim
2980     }
2981     c
2982     < {
2983       \vrule height \c_zero_dim depth \box_dp:N \@@arstrutbox width \c_zero_dim
2984       \end { minipage }
2985       \@@_cell_end:
2986     }
2987   }

```

We test for the presence of a <.

```

2988   \@@_make_m_preamble_x:n
2989 }

```

For w and W

```

2990 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2991 {
2992   \tl_gput_right:Nn \g_@@_preamble_tl
2993   {
2994     > {
2995       \dim_set:Nn \l_@@_col_width_dim { #4 }
2996       \hbox_set:Nw \l_@@_cell_box
2997       \@@_cell_begin:
2998       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2999     }
3000     c
3001     < {
3002       \@@_cell_end:
3003       \hbox_set_end:
3004       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3005       #1
3006       \@@_adjust_size_box:
3007       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3008     }
3009   }

```

We test for the presence of a <.

```
3010 \@@_make_m_preamble_x:n
3011 }
```

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

```
3012 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3013 {
3014   \str_if_eq:nnTF { #1 } { < }
3015     { \@@_make_m_preamble_ix:n }
3016     { \@@_make_m_preamble:n { #1 } }
3017 }
3018 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3019 {
3020   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3021   \@@_make_m_preamble_x:n
3022 }
```

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`).

```
3023 \cs_new_protected:Npn \@@_put_box_in_flow:
3024 {
3025   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3026   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3027   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3028     { \box_use_drop:N \l_tmpa_box }
3029     { \@@_put_box_in_flow_i: }
3030 }
```

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

```
3031 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3032 {
3033   \pgfpicture
3034     \@@_qpoint:n { row - 1 }
3035     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3036     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3037     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3038     \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).

```
3039   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3040     {
3041       \int_set:Nn \l_tmpa_int
3042         {
3043           \str_range:Nnn
3044             \l_@@_baseline_tl
3045             6
3046             { \tl_count:o \l_@@_baseline_tl }
3047         }
3048       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3049     }
3050   {
3051     \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3052     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3053     {
3054       \str_if_eq:onTF \l_@@_baseline_tl { b }
3055       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3056       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
```



```

3057     }
3058     \bool_lazy_or:nnT
3059     { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3060     { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3061     {
3062         \@@_error:n { bad-value-for-baseline }
3063         \int_set_eq:NN \l_tmpa_int \c_one_int
3064     }
3065     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3066     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3067 }
3068 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3069 \endpgfpicture
3070 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3071 \box_use_drop:N \l_tmpa_box
3072 }

```

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).

```

3073 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3074 {

```

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.

```

3075     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3076     {
3077         \int_compare:nNnT { \c@jCol } > { \c_one_int }
3078         {
3079             \box_set_wd:Nn \l_@@_the_array_box
3080             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3081         }
3082     }

```

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

```

3083     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3084     \bool_if:NT \l_@@_caption_above_bool
3085     {
3086         \tl_if_empty:NF \l_@@_caption_tl
3087         {
3088             \bool_set_false:N \g_@@_caption_finished_bool
3089             \int_gzero:N \c@tabularnote
3090             \@@_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).

```

3091         \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3092         {
3093             \tl_gput_right:Ne \g_@@_aux_tl
3094             {
3095                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3096                 { \int_use:N \g_@@_notes_caption_int }
3097             }
3098             \int_gzero:N \g_@@_notes_caption_int
3099         }
3100     }
3101 }

```

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

```

3102   \hbox
3103   {
3104   \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.

```

3105   \@@_create_extra_nodes:
3106   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3107 }

```

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).

```

3108   \bool_lazy_any:nT
3109   {
3110     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3111     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3112     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3113   }
3114   \@@_insert_tabularnotes:
3115   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3116   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3117   \end { minipage }
3118 }

```

```

3119 \cs_new_protected:Npn \@@_insert_caption:
3120 {
3121   \tl_if_empty:NF \l_@@_caption_tl
3122   {
3123     \cs_if_exist:NTF \capttype
3124     { \@@_insert_caption_i: }
3125     { \@@_error:n { caption~outside~float } }
3126   }
3127 }

```

```

3128 \cs_new_protected:Npn \@@_insert_caption_i:
3129 {
3130   \group_begin:

```

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

```

3131   \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.

```

3132   \IfPackageLoadedT { floatrow }
3133   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3134   \tl_if_empty:NTF \l_@@_short_caption_tl
3135   { \caption }
3136   { \caption [ \l_@@_short_caption_tl ] }
3137   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3138 \bool_if:NF \g_@@_caption_finished_bool
3139 {
3140   \bool_gset_true:N \g_@@_caption_finished_bool
3141   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3142   \int_gzero:N \c@tabularnote
3143 }
3144 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3145 \group_end:
3146 }
3147 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3148 {
3149   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3150   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3151 }
3152 \cs_new_protected:Npn \@@_insert_tabularnotes:
3153 {
3154   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3155   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3156   \skip_vertical:N 0.65ex

```

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

```

3157 \group_begin:
3158 \l_@@_notes_code_before_tl
3159 \tl_if_empty:NF \g_@@_tabularnote_tl
3160 {
3161   \g_@@_tabularnote_tl \par
3162   \tl_gclear:N \g_@@_tabularnote_tl
3163 }

```

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.

```

3164 \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3165 {
3166   \bool_if:NTF \l_@@_notes_para_bool
3167   {
3168     \begin { tabularnotes* }
3169     \seq_map_inline:Nn \g_@@_notes_seq
3170     { \@@_one_tabularnote:nn ##1 }
3171     \strut
3172     \end { tabularnotes* }

```

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

```

3173 \par
3174 }
3175 {
3176   \tabularnotes
3177   \seq_map_inline:Nn \g_@@_notes_seq
3178   { \@@_one_tabularnote:nn ##1 }
3179   \strut
3180   \endtabularnotes
3181 }
3182 }
3183 \unskip
3184 \group_end:
3185 \bool_if:NT \l_@@_notes_bottomrule_bool
3186 {
3187   \IfPackageLoadedTF { booktabs }
3188   {

```

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

```

3189 \skip_vertical:N \aboverulesep

```

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

```

3190         { \CT@arc@ \hrule height \heavyrulewidth }
3191     }
3192     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3193 }
3194 \l_@@_notes_code_after_tl
3195 \seq_gclear:N \g_@@_notes_seq
3196 \seq_gclear:N \g_@@_notes_in_caption_seq
3197 \int_gzero:N \c@tabularnote
3198 }

```

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.

```

3199 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3200 {
3201     \tl_if_novalue:nTF { #1 }
3202     { \item }
3203     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3204 }

```

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.

```

3205 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3206 {
3207     \pgfpicture
3208     \@@_qpoint:n { row - 1 }
3209     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3210     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3211     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3212     \endpgfpicture
3213     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3214     \int_if_zero:nT { \l_@@_first_row_int }
3215     {
3216         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3217         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3218     }
3219     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3220 }

```

Now, the general case.

```

3221 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3222 {

```

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

```

3223     \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3224     { \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`.

```

3225     \pgfpicture
3226     \@@_qpoint:n { row - 1 }
3227     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3228     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3229     {
3230         \int_set:Nn \l_tmpa_int
3231         {
3232             \str_range:Nnn
3233             \l_@@_baseline_tl
3234             { 6 }
3235             { \tl_count:o \l_@@_baseline_tl }

```

```

3236     }
3237     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3238   }
3239   {
3240     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3241     \bool_lazy_or:nnT
3242       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3243       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3244     {
3245       \@@_error:n { bad-value-for~baseline }
3246       \int_set:Nn \l_tmpa_int 1
3247     }
3248     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3249   }
3250   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3251   \endpgfpicture
3252   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3253   \int_if_zero:nT { \l_@@_first_row_int }
3254   {
3255     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3256     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3257   }
3258   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3259 }

```

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.

```

3260 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3261 {

```

We will compute the real width of both delimiters used.

```

3262   \dim_zero_new:N \l_@@_real_left_delim_dim
3263   \dim_zero_new:N \l_@@_real_right_delim_dim
3264   \hbox_set:Nn \l_tmpb_box
3265   {
3266     \m@th
3267     \c_math_toggle_token
3268     \left #1
3269     \vcenter
3270     {
3271       \vbox_to_ht:nn
3272         { \box_ht_plus_dp:N \l_tmpa_box }
3273       { }
3274     }
3275     \right .
3276     \c_math_toggle_token
3277   }
3278   \dim_set:Nn \l_@@_real_left_delim_dim
3279   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3280   \hbox_set:Nn \l_tmpb_box
3281   {
3282     \m@th
3283     \c_math_toggle_token
3284     \left .
3285     \vbox_to_ht:nn
3286       { \box_ht_plus_dp:N \l_tmpa_box }
3287     { }
3288     \right #2
3289     \c_math_toggle_token
3290   }
3291   \dim_set:Nn \l_@@_real_right_delim_dim
3292   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3293     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3294     \@@_put_box_in_flow:
3295     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3296 }

```

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).

```

3297 \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).

```

3298 {
3299   \peek_remove_spaces:n
3300   {
3301     \peek_meaning:NTF \end
3302     { \@@_analyze_end:Nn }
3303     {
3304       \@@_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`).

```

3305       \@@_array:o \g_@@_array_preamble_tl
3306     }
3307   }
3308 }
3309 {
3310   \@@_create_col_nodes:
3311   \endarray
3312 }

```

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

```

3313 \NewDocumentEnvironment { @@-light-syntax } { b }
3314 {

```

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`.

```

3315   \tl_if_empty:nT { #1 }
3316   { \@@_fatal:n { empty-environment } }
3317   \tl_if_in:nnT { #1 } { & }
3318   { \@@_fatal:n { ampersand-in-light-syntax } }
3319   \tl_if_in:nnT { #1 } { \ }
3320   { \@@_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`.

```

3321   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3322 }

```

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.

```

3323 {
3324   \@@_create_col_nodes:
3325   \endarray
3326 }

3327 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3328 {
3329   \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).

```

3330   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3331   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3332   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3333     { \seq_set_split:Nee }
3334     { \seq_set_split:Non }
3335   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3336   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3337   \tl_if_empty:NF \l_tmpa_tl
3338   { \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.

```

3339   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3340     { \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`).

```

3341   \tl_build_begin:N \l_@@_new_body_tl
3342   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3343   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3344   \@@_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).

```

3345   \seq_map_inline:Nn \l_@@_rows_seq
3346   {
3347     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3348     \@@_line_with_light_syntax:n { ##1 }
3349   }
3350   \tl_build_end:N \l_@@_new_body_tl

3351   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3352   {
3353     \int_set:Nn \l_@@_last_col_int
3354       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3355   }

```

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.

```

3356   \@@_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`).

```

3357   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3358 }

```

```

3359 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3360 {
3361   \seq_clear_new:N \l_@@_cells_seq
3362   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3363   \int_set:Nn \l_@@_nb_cols_int
3364   {
3365     \int_max:nn
3366     { \l_@@_nb_cols_int }
3367     { \seq_count:N \l_@@_cells_seq }
3368   }
3369   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3370   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3371   \seq_map_inline:Nn \l_@@_cells_seq
3372   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3373 }
3374 \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`.

```

3375 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3376 {
3377   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3378   { \@@_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.

```

3379   \end { #2 }
3380 }

```

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`).

```

3381 \cs_new:Npn \@@_create_col_nodes:
3382 {
3383   \crrr
3384   \int_if_zero:nT { \l_@@_first_col_int }
3385   {
3386     \omit
3387     \hbox_overlap_left:n
3388     {
3389       \bool_if:NT \l_@@_code_before_bool
3390       { \pgfsys@markposition { \@@_env: - col - 0 } }
3391       \pgfpicture
3392       \pgfrememberpicturepositiononpagetrue
3393       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3394       \str_if_empty:NF \l_@@_name_str
3395       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3396       \endpgfpicture
3397       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3398     }
3399     &
3400   }
3401   \omit

```

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

```

3402   \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`).

```

3403   \int_if_zero:nTF { \l_@@_first_col_int }
3404   {
3405     \@@_mark_position:n { 1 }

```



```

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

```

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.

```

3431 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3432 \bool_if:NF \l_@@_auto_columns_width_bool
3433 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3434 {
3435 \bool_lazy_and:nnTF
3436 { \l_@@_auto_columns_width_bool }
3437 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3438 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3439 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3440 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3441 }
3442 \skip_horizontal:N \g_tmpa_skip
3443 \hbox
3444 {
3445 \@@_mark_position:n { 2 }
3446 \pgfpicture
3447 \pgfrememberpicturepositiononpagetrue
3448 \pgfcoordinate { \@@_env: - col - 2 }
3449 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3450 \@@_node_alias:n { 2 }
3451 \endpgfpicture
3452 }

```

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.

```

3453 \int_gset_eq:NN \g_tmpa_int \c_one_int
3454 \bool_if:NTF \g_@@_last_col_found_bool
3455 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3456 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3457 {
3458 &
3459 \omit

```

```
3460 \int_gincr:N \g_tmpa_int
```

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

```
3461 \skip_horizontal:N \g_tmpa_skip
3462 \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
3463 \pgfpicture
3464 \pgfrememberpicturepositiononpagetrue
3465 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3466 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3467 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3468 \endpgfpicture
3469 }
```

```
3470 &
3471 \omit
```

If there is only one column (and a potential “last column”), we don’t have to put the following code (there is only one column and we have put the correct code previously).

```
3472 \bool_lazy_or:nnF
3473 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3474 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3475 {
3476 \skip_horizontal:N \g_tmpa_skip
3477 \int_gincr:N \g_tmpa_int
3478 \bool_lazy_any:nF
3479 {
3480 \g_@@_delims_bool
3481 \l_@@_tabular_bool
3482 { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3483 \l_@@_exterior_arraycolsep_bool
3484 \l_@@_bar_at_end_of_pream_bool
3485 }
3486 { \skip_horizontal:n { - \col@sep } }
3487 \bool_if:NT \l_@@_code_before_bool
3488 {
3489 \hbox
3490 {
3491 \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.

```
3492 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3493 { \skip_horizontal:n { - \arraycolsep } }
3494 \pgfsys@markposition
3495 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3496 \skip_horizontal:n { 0.5 \arrayrulewidth }
3497 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3498 { \skip_horizontal:N \arraycolsep }
3499 }
3500 }
3501 \pgfpicture
3502 \pgfrememberpicturepositiononpagetrue
3503 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3504 {
3505 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3506 {
3507 \pgfpoint
3508 { - 0.5 \arrayrulewidth - \arraycolsep }
3509 \c_zero_dim
3510 }
3511 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
```

```

3512     }
3513     \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3514 \endpgfpicture
3515 }

3516 \bool_if:NT \g_@@_last_col_found_bool
3517 {
3518     \hbox_overlap_right:n
3519     {
3520         \skip_horizontal:N \g_@@_width_last_col_dim
3521         \skip_horizontal:N \col@sep
3522         \bool_if:NT \l_@@_code_before_bool
3523         {
3524             \pgfsys@markposition
3525             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3526         }
3527         \pgfpicture
3528         \pgfrememberpicturepositiononpagetrue
3529         \pgfcoordinate
3530         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3531         \pgfpointorigin
3532         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3533         \endpgfpicture
3534     }
3535 }
3536 % \cr
3537 }

3538 \cs_new_protected:Npn \@@_mark_position:n #1
3539 {
3540     \bool_if:NT \l_@@_code_before_bool
3541     {
3542         \hbox
3543         {
3544             \skip_horizontal:n { -0.5 \arrayrulewidth }
3545             \pgfsys@markposition { \@@_env: - col - #1 }
3546             \skip_horizontal:n { 0.5 \arrayrulewidth }
3547         }
3548     }
3549 }

3550 \cs_new_protected:Npn \@@_node_alias:n #1
3551 {
3552     \str_if_empty:NF \l_@@_name_str
3553     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3554 }

```

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

```

3555 \tl_const:Nn \c_@@_preamble_first_col_tl
3556 {
3557     >
3558     {

```

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

```

3559     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3560     \bool_gset_true:N \g_@@_after_col_zero_bool
3561     \@@_begin_of_row:
3562     \hbox_set:Nw \l_@@_cell_box
3563     \@@_math_toggle:
3564     \@@_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”.

```

3565     \int_compare:nNt { \c@iRow } > { \c_zero_int }
3566     {
3567         \bool_lazy_or:nnT
3568         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3569         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3570         {
3571             \l_@@_code_for_first_col_tl
3572             \xglobal \colorlet { nicematrix-first-col } { . }
3573         }
3574     }
3575 }

```

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`.

```

3576     l
3577     <
3578     {
3579         \@@_math_toggle:
3580         \hbox_set_end:
3581         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3582         \@@_adjust_size_box:
3583         \@@_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.

```

3584     \dim_gset:Nn \g_@@_width_first_col_dim
3585     { \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.

```

3586     \hbox_overlap_left:n
3587     {
3588         \dim_compare:nNtF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3589         { \@@_node_cell: }
3590         { \box_use_drop:N \l_@@_cell_box }
3591         \skip_horizontal:N \l_@@_left_delim_dim
3592         \skip_horizontal:N \l_@@_left_margin_dim
3593         \skip_horizontal:N \l_@@_extra_left_margin_dim
3594     }
3595     \bool_gset_false:N \g_@@_empty_cell_bool
3596     \skip_horizontal:n { -2 \col@sep }
3597 }
3598 }

```

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

```

3599 \tl_const:Nn \c_@@_preamble_last_col_tl
3600 {
3601     >
3602     {
3603         \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).

```

3604     \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.

```

3605     \bool_gset_true:N \g_@@_last_col_found_bool
3606     \int_gincr:N \c@jCol
3607     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3608     \hbox_set:Nw \l_@@_cell_box
3609     \@@_math_toggle:
3610     \@@_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”.

```

3611     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3612     {
3613         \bool_lazy_or:nnT
3614         { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3615         { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3616         {
3617             \l_@@_code_for_last_col_tl
3618             \xglobal \colorlet { nicematrix-last-col } { . }
3619         }
3620     }
3621 }
3622 1
3623 <
3624 {
3625     \@@_math_toggle:
3626     \hbox_set_end:
3627     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3628     \@@_adjust_size_box:
3629     \@@_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.

```

3630     \dim_gset:Nn \g_@@_width_last_col_dim
3631     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3632     \skip_horizontal:n { -2 \col@sep }

```

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

```

3633     \hbox_overlap_right:n
3634     {
3635         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3636         {
3637             \skip_horizontal:N \l_@@_right_delim_dim
3638             \skip_horizontal:N \l_@@_right_margin_dim
3639             \skip_horizontal:N \l_@@_extra_right_margin_dim
3640             \@@_node_cell:
3641         }
3642     }
3643     \bool_gset_false:N \g_@@_empty_cell_bool
3644 }
3645 }

```

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

```

3646 \NewDocumentEnvironment { NiceArray } { }
3647 {
3648     \bool_gset_false:N \g_@@_delims_bool
3649     \str_if_empty:NT \g_@@_name_env_str
3650     { \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).

```

3651     \NiceArrayWithDelims . .
3652 }
3653 { \endNiceArrayWithDelims }

```

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

```

3654 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3655 {
3656     \NewDocumentEnvironment { #1 NiceArray } { }
3657     {

```

```

3658 \bool_gset_true:N \g_@@_delims_bool
3659 \str_if_empty:NT \g_@@_name_env_str
3660 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3661 \@@_test_if_math_mode:
3662 \NiceArrayWithDelims #2 #3
3663 }
3664 { \endNiceArrayWithDelims }
3665 }
3666 \@@_def_env:NNN p ( )
3667 \@@_def_env:NNN b [ ]
3668 \@@_def_env:NNN B \{ \}
3669 \@@_def_env:NNN v \vert \vert
3670 \@@_def_env:NNN V \Vert \Vert

```

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

```

3671 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3672 {
3673   \bool_set_false:N \l_@@_preamble_bool
3674   \tl_clear:N \l_tmpa_tl
3675   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3676   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3677   \tl_put_right:Nn \l_tmpa_tl
3678   {
3679     *
3680     {
3681       \int_case:nnF \l_@@_last_col_int
3682       {
3683         { -2 } { \c@MaxMatrixCols }
3684         { -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).

```

3685   }
3686   { \int_eval:n { \l_@@_last_col_int - 1 } }
3687 }
3688 { #2 }
3689 }
3690 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3691 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3692 }
3693 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3694 \clist_map_inline:nn { p , b , B , v , V }
3695 {
3696   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3697   {
3698     \bool_gset_true:N \g_@@_delims_bool
3699     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3700     \int_if_zero:nT { \l_@@_last_col_int }
3701     {
3702       \bool_set_true:N \l_@@_last_col_without_value_bool
3703       \int_set:Nn \l_@@_last_col_int { -1 }
3704     }
3705     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3706     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3707   }
3708   { \use:c { end #1 NiceArray } }
3709 }

```

We define also an environment `{NiceMatrix}`

```

3710 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3711 {
3712   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3713   \int_if_zero:nT { \l_@@_last_col_int }
3714   {
3715     \bool_set_true:N \l_@@_last_col_without_value_bool
3716     \int_set:Nn \l_@@_last_col_int { -1 }
3717   }
3718   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3719   \bool_lazy_or:nnT
3720     { \clist_if_empty_p:N \l_@@_vlines_clist }
3721     { \l_@@_except_borders_bool }
3722     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3723   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3724 }
3725 { \endNiceArray }

```

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

```

3726 \cs_new_protected:Npn \@@_NotEmpty:
3727 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3728 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3729 {

```

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.

```

3730   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3731     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3732   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3733   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3734   \tl_if_empty:NF \l_@@_short_caption_tl
3735   {
3736     \tl_if_empty:NT \l_@@_caption_tl
3737     {
3738       \@@_error_or_warning:n { short-caption-without-caption }
3739       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3740     }
3741   }
3742   \tl_if_empty:NF \l_@@_label_tl
3743   {
3744     \tl_if_empty:NT \l_@@_caption_tl
3745     { \@@_error_or_warning:n { label-without-caption } }
3746   }
3747   \NewDocumentEnvironment { TabularNote } { b }
3748   {
3749     \bool_if:NTF \l_@@_in_code_after_bool
3750       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3751       {
3752         \tl_if_empty:NF \g_@@_tabularnote_tl
3753         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3754         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3755       }
3756   }
3757   { }
3758   \@@_settings_for_tabular:
3759   \NiceArray { #2 }
3760 }
3761 { \endNiceArray }
3762 \cs_new_protected:Npn \@@_settings_for_tabular:
3763 {

```

```

3764 \bool_set_true:N \l_@@_tabular_bool
3765 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3766 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3767 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3768 }

3769 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3770 {
3771   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3772   \dim_set:Nn \l_@@_width_dim { #1 }
3773   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3774   \@@_settings_for_tabular:
3775   \NiceArray { #3 }
3776 }
3777 {
3778   \endNiceArray
3779   \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3780     { \@@_error:n { NiceTabularX~without~X } }
3781 }

3782 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3783 {
3784   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3785   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3786   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3787   \@@_settings_for_tabular:
3788   \NiceArray { #3 }
3789 }
3790 { \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`).

```

3791 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3792 {
3793   \bool_lazy_all:nT
3794   {
3795     { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3796     { \l_@@_hvlines_bool }
3797     { ! \g_@@_delims_bool }
3798     { ! \l_@@_except_borders_bool }
3799   }
3800   {
3801     \bool_set_true:N \l_@@_except_borders_bool
3802     \clist_if_empty:NF \l_@@_corners_clist
3803     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3804     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3805     {
3806       \@@_stroke_block:nnn
3807       {
3808         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3809         draw = \l_@@_rules_color_tl
3810       }
3811       { 1-1 }
3812       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3813     }
3814   }
3815 }

```



```

3816 \cs_new_protected:Npn \@@_after_array:
3817 {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3818 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3819 \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.

```

3820 \bool_if:NT \g_@@_last_col_found_bool
3821 { \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`.

```

3822 \bool_if:NT \l_@@_last_col_without_value_bool
3823 { \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.

```

3824 \bool_if:NT \l_@@_last_row_without_value_bool
3825 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3826 \tl_gput_right:Ne \g_@@_aux_tl
3827 {
3828   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3829   {
3830     \int_use:N \l_@@_first_row_int ,
3831     \int_use:N \c@iRow ,
3832     \int_use:N \g_@@_row_total_int ,
3833     \int_use:N \l_@@_first_col_int ,
3834     \int_use:N \c@jCol ,
3835     \int_use:N \g_@@_col_total_int
3836   }
3837 }

```

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`).

```

3838 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3839 {
3840   \tl_gput_right:Ne \g_@@_aux_tl
3841   {
3842     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3843     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3844   }
3845 }
3846 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3847 {
3848   \tl_gput_right:Ne \g_@@_aux_tl
3849   {
3850     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3851     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3852     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3853     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3854   }
3855 }

```

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

```
3856 \@@_create_diag_nodes:
```

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

```
3857 \pgfpicture
3858 \@@_create_aliases_last:
3859 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3860 \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.

```
3861 \bool_if:NT \l_@@_parallelize_diags_bool
3862 {
3863   \int_gzero:N \g_@@_ddots_int
3864   \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.

```
3865   \dim_gzero:N \g_@@_delta_x_one_dim
3866   \dim_gzero:N \g_@@_delta_y_one_dim
3867   \dim_gzero:N \g_@@_delta_x_two_dim
3868   \dim_gzero:N \g_@@_delta_y_two_dim
3869 }
3870 \bool_set_false:N \l_@@_initial_open_bool
3871 \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.

```
3872 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
3873 \@@_draw_dotted_lines:
```

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

```
3874 \clist_if_empty:NF \l_@@_corners_clist
3875 {
3876   \bool_if:NTF \l_@@_no_cell_nodes_bool
3877   { \@@_error:n { corners~with~no~cell~nodes } }
3878   { \@@_compute_corners: }
3879 }
```

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

```
3880 \@@_adjust_pos_of_blocks_seq:
3881 \@@_deal_with_rounded_corners:
3882 \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3883 \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

---

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

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

```

3884 \IfPackageLoadedT { tikz }
3885 {
3886   \tikzset
3887   {
3888     every-picture / .style =
3889     {
3890       overlay ,
3891       remember-picture ,
3892       name-prefix = \@@_env: -
3893     }
3894   }
3895 }
3896 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3897 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3898 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3899 \cs_set_eq:NN \OverBrace \@@_OverBrace
3900 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3901 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3902 \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.

```

3903 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3904 \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.

```

3905 \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).

```

3906 \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`).

```

3907 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3908 { \@@_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:`.

```

3909 \bool_set_true:N \l_@@_in_code_after_bool
3910 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3911 \scan_stop:
3912 \tl_gclear:N \g_nicematrix_code_after_tl
3913 \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.

```

3914 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3915 \tl_if_empty:NF \g_@@_pre_code_before_tl
3916 {
3917   \tl_gput_right:Ne \g_@@_aux_tl
3918   {
3919     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3920     { \exp_not:o \g_@@_pre_code_before_tl }
3921   }
3922   \tl_gclear:N \g_@@_pre_code_before_tl
3923 }

```

```

3924 \tl_if_empty:NF \g_nicematrix_code_before_tl
3925 {
3926   \tl_gput_right:Ne \g_@@_aux_tl
3927   {
3928     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3929     { \exp_not:o \g_nicematrix_code_before_tl }
3930   }
3931   \tl_gclear:N \g_nicematrix_code_before_tl
3932 }

3933 \str_gclear:N \g_@@_name_env_str
3934 \@@_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`.

```

3935 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3936 }

3937 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
3938 {
3939   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3940   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

3941   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3942   { 0.6 \l_@@_xdots_shorten_start_dim }
3943   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3944   { 0.6 \l_@@_xdots_shorten_end_dim }
3945 }

```

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`.

```

3946 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3947 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

3948 \cs_new_protected:Npn \@@_create_alias_nodes:
3949 {
3950   \int_step_inline:nn { \c@iRow }
3951   {
3952     \pgfnodealias
3953     { \l_@@_name_str - ##1 - last }
3954     { \@@_env: - ##1 - \int_use:N \c@jCol }
3955   }
3956   \int_step_inline:nn { \c@jCol }
3957   {
3958     \pgfnodealias
3959     { \l_@@_name_str - last - ##1 }
3960     { \@@_env: - \int_use:N \c@iRow - ##1 }
3961   }
3962   \pgfnodealias
3963   { \l_@@_name_str - last - last }
3964   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3965 }

```

---

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

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format  $i-j$ . However, the user is allowed to omit  $i$  or  $j$  (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

3966 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3967 {
3968   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3969   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3970 }

```

The following command must *not* be protected.

```

3971 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3972 {
3973   { #1 }
3974   { #2 }
3975   {
3976     \int_compare:nNnTF { #3 } > { 98 }
3977     { \int_use:N \c@iRow }
3978     { #3 }
3979   }
3980   {
3981     \int_compare:nNnTF { #4 } > { 98 }
3982     { \int_use:N \c@jCol }
3983     { #4 }
3984   }
3985   { #5 }
3986 }

```

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).

```

3987 \hook_gput_code:nnn { begindocument } { . }
3988 {
3989   \cs_new_protected:Npe \@@_draw_dotted_lines:
3990   {
3991     \c_@@_pgfortikzpicture_tl
3992     \@@_draw_dotted_lines_i:
3993     \c_@@_endpgfortikzpicture_tl
3994   }
3995 }

```

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

```

3996 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3997 {
3998   \pgfrememberpicturepositiononpagetrue
3999   \pgf@relevantforpicturesizefalse
4000   \g_@@_HVdotsfor_lines_tl
4001   \g_@@_Vdots_lines_tl
4002   \g_@@_Ddots_lines_tl
4003   \g_@@_Idots_lines_tl
4004   \g_@@_Cdots_lines_tl
4005   \g_@@_Ldots_lines_tl
4006 }

4007 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4008 {
4009   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4010   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4011 }

```

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

```

4012 \pgfdeclareshape { @@_diag_node }
4013 {
4014   \savedanchor { \five }
4015   {
4016     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4017     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4018   }
4019   \anchor { 5 } { \five }
4020   \anchor { center } { \pgfpointorigin }
4021   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4022   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4023   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4024   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4025   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4026   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4027   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4028   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4029   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4030   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4031 }

```

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).

```

4032 \cs_new_protected:Npn \@@_create_diag_nodes:
4033 {
4034   \pgfpicture
4035   \pgfrememberpicturepositiononpagetrue
4036   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4037   {
4038     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4039     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4040     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4041     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4042     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4043     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4044     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4045     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4046     \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.

```

4047     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4048     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4049     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4050     \str_if_empty:NF \l_@@_name_str
4051     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4052   }

```

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

```

4053     \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4054     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4055     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4056     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4057     \pgfcoordinate
4058     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4059     \pgfnodealias
4060     { \@@_env: - last }
4061     { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4062     \str_if_empty:NF \l_@@_name_str
4063     {

```

```

4064     \pgfnodealias
4065     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4066     { \@@_env: - \int_use:N \l_tmpa_int }
4067     \pgfnodealias
4068     { \l_@@_name_str - last }
4069     { \@@_env: - last }
4070   }
4071 \endpgfpicture
4072 }

```

## 16 We draw the dotted lines

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

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

The command `\@@_find_extremities_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.

```

4073 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4074 {

```

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

```

4075   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4076   \int_set:Nn \l_@@_initial_i_int { #1 }
4077   \int_set:Nn \l_@@_initial_j_int { #2 }
4078   \int_set:Nn \l_@@_final_i_int { #1 }
4079   \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.

```

4080   \bool_set_false:N \l_@@_stop_loop_bool
4081   \bool_do_until:Nn \l_@@_stop_loop_bool
4082   {
4083     \int_add:Nn \l_@@_final_i_int { #3 }
4084     \int_add:Nn \l_@@_final_j_int { #4 }
4085     \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.

```

4086     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4087     \if_int_compare:w #3 = \c_one_int
4088     \bool_set_true:N \l_@@_final_open_bool
4089     \else:
4090     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4091     \bool_set_true:N \l_@@_final_open_bool
4092     \fi:
4093     \fi:
4094     \else:
4095     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4096     \if_int_compare:w #4 = -1
4097     \bool_set_true:N \l_@@_final_open_bool
4098     \fi:
4099     \else:
4100     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4101     \if_int_compare:w #4 = \c_one_int
4102     \bool_set_true:N \l_@@_final_open_bool
4103     \fi:
4104     \fi:
4105     \fi:
4106     \fi:
4107     \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.

```

4108     {

```

We do a step backwards.

```

4109         \int_sub:Nn \l_@@_final_i_int { #3 }
4110         \int_sub:Nn \l_@@_final_j_int { #4 }
4111         \bool_set_true:N \l_@@_stop_loop_bool
4112     }

```

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`.

```

4113     {
4114         \cs_if_exist:cTF
4115         {
4116             @@ _ dotted _
4117             \int_use:N \l_@@_final_i_int -
4118             \int_use:N \l_@@_final_j_int
4119         }
4120         {
4121             \int_sub:Nn \l_@@_final_i_int { #3 }
4122             \int_sub:Nn \l_@@_final_j_int { #4 }
4123             \bool_set_true:N \l_@@_final_open_bool
4124             \bool_set_true:N \l_@@_stop_loop_bool
4125         }
4126         {
4127             \cs_if_exist:cTF
4128             {
4129                 pgf @ sh @ ns @ \@@_env:
4130                 - \int_use:N \l_@@_final_i_int
4131                 - \int_use:N \l_@@_final_j_int
4132             }
4133             { \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.

```

4134     {

```



```

4135         \cs_set_nopar:cpn
4136         {
4137             @@ _ dotted _
4138             \int_use:N \l_@@_final_i_int -
4139             \int_use:N \l_@@_final_j_int
4140         }
4141         { }
4142     }
4143 }
4144 }
4145 }

```

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

```

4146     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4147     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4148     \bool_do_until:Nn \l_@@_stop_loop_bool
4149     {
4150         \int_sub:Nn \l_@@_initial_i_int { #3 }
4151         \int_sub:Nn \l_@@_initial_j_int { #4 }
4152         \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.

```

4153         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4154         \if_int_compare:w #3 = \c_one_int
4155             \bool_set_true:N \l_@@_initial_open_bool
4156         \else:

```

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

```

4157         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4158             \bool_set_true:N \l_@@_initial_open_bool
4159         \fi:
4160     \fi:
4161 \else:
4162     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4163         \if_int_compare:w #4 = \c_one_int
4164             \bool_set_true:N \l_@@_initial_open_bool
4165         \fi:
4166     \else:
4167         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4168             \if_int_compare:w #4 = -1
4169                 \bool_set_true:N \l_@@_initial_open_bool
4170             \fi:
4171         \fi:
4172     \fi:
4173 \fi:
4174 \bool_if:NTF \l_@@_initial_open_bool
4175 {
4176     \int_add:Nn \l_@@_initial_i_int { #3 }
4177     \int_add:Nn \l_@@_initial_j_int { #4 }
4178     \bool_set_true:N \l_@@_stop_loop_bool
4179 }
4180 {
4181     \cs_if_exist:cTF
4182     {
4183         @@ _ dotted _
4184         \int_use:N \l_@@_initial_i_int -
4185         \int_use:N \l_@@_initial_j_int
4186     }

```

```

4187     {
4188         \int_add:Nn \l_@@_initial_i_int { #3 }
4189         \int_add:Nn \l_@@_initial_j_int { #4 }
4190         \bool_set_true:N \l_@@_initial_open_bool
4191         \bool_set_true:N \l_@@_stop_loop_bool
4192     }
4193     {
4194         \cs_if_exist:cTF
4195         {
4196             pgf @ sh @ ns @ \@@_env:
4197             - \int_use:N \l_@@_initial_i_int
4198             - \int_use:N \l_@@_initial_j_int
4199         }
4200         { \bool_set_true:N \l_@@_stop_loop_bool }
4201         {
4202             \cs_set_nopar:cpn
4203             {
4204                 @@ _ dotted _
4205                 \int_use:N \l_@@_initial_i_int -
4206                 \int_use:N \l_@@_initial_j_int
4207             }
4208             { }
4209         }
4210     }
4211 }
4212 }

```

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.

```

4213 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4214 {
4215     { \int_use:N \l_@@_initial_i_int }

```

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

```

4216     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4217     { \int_use:N \l_@@_final_i_int }
4218     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4219     { }
4220 }
4221 }

```

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

```

4222 \cs_new_protected:Npn \@@_open_shorten:
4223 {
4224     \bool_if:NT \l_@@_initial_open_bool
4225     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4226     \bool_if:NT \l_@@_final_open_bool
4227     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4228 }

```

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).

```

4229 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4230 {
4231     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4232 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4233 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4234 \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`.

```

4235 \seq_if_empty:NF \g_@@_submatrix_seq
4236 {
4237   \seq_map_inline:Nn \g_@@_submatrix_seq
4238   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4239 }
4240 }

```

**#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.

```

4241 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4242 {
4243   \if_int_compare:w #3 > #1
4244   \else:
4245     \if_int_compare:w #1 > #5
4246     \else:
4247       \if_int_compare:w #4 > #2
4248       \else:
4249         \if_int_compare:w #2 > #6
4250         \else:
4251           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4252           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4253           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4254           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4255         \fi:
4256       \fi:
4257     \fi:
4258   \fi:
4259 }

4260 \cs_new_protected:Npn \@@_set_initial_coords:
4261 {
4262   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4263   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4264 }
4265 \cs_new_protected:Npn \@@_set_final_coords:
4266 {

```

```

4267 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4268 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4269 }
4270 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4271 {
4272   \pgfpointanchor
4273   {
4274     \@@_env:
4275     - \int_use:N \l_@@_initial_i_int
4276     - \int_use:N \l_@@_initial_j_int
4277   }
4278   { #1 }
4279   \@@_set_initial_coords:
4280 }
4281 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4282 {
4283   \pgfpointanchor
4284   {
4285     \@@_env:
4286     - \int_use:N \l_@@_final_i_int
4287     - \int_use:N \l_@@_final_j_int
4288   }
4289   { #1 }
4290   \@@_set_final_coords:
4291 }
4292 \cs_new_protected:Npn \@@_open_x_initial_dim:
4293 {
4294   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4295   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4296   {
4297     \cs_if_exist:cT
4298     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4299     {
4300       \pgfpointanchor
4301       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4302       { west }
4303       \dim_set:Nn \l_@@_x_initial_dim
4304       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4305     }
4306   }

```

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

```

4307 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4308 {
4309   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4310   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4311   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4312 }
4313 }
4314 \cs_new_protected:Npn \@@_open_x_final_dim:
4315 {
4316   \dim_set:Nn \l_@@_x_final_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_@@_final_j_int }
4321     {
4322       \pgfpointanchor
4323       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4324       { east }
4325       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4326       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4327     }

```

```
4328     }
```

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

```
4329     \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4330     {
4331         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4332         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4333         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4334     }
4335 }
```

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.

```
4336 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4337 {
4338     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4339     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4340     {
4341         \@@_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.

```
4342     \group_begin:
4343     \@@_open_shorten:
4344     \int_if_zero:nTF { #1 }
4345     { \color { nicematrix-first-row } }
4346     {
```

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.

```
4347         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4348         { \color { nicematrix-last-row } }
4349     }
4350     \keys_set:nn { nicematrix / xdots } { #3 }
4351     \@@_color:o \l_@@_xdots_color_tl
4352     \@@_actually_draw_Ldots:
4353     \group_end:
4354 }
4355 }
```

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`.

```
4356 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4357 {
4358     \bool_if:NTF \l_@@_initial_open_bool
4359     {
4360         \@@_open_x_initial_dim:
4361         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4362         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4363     }
4364     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4365 \bool_if:NTF \l_@@_final_open_bool
4366 {
4367   \@@_open_x_final_dim:
4368   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4369   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4370 }
4371 { \@@_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.

```

4372 \bool_lazy_all:nTF
4373 {
4374   \l_@@_initial_open_bool
4375   \l_@@_final_open_bool
4376   { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4377 }
4378 {
4379   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4380   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4381 }

```

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

```

4382 {
4383   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4384   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4385 }
4386 \@@_draw_line:
4387 }

```

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.

```

4388 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4389 {
4390   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4391   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4392   {
4393     \@@_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.

```

4394 \group_begin:
4395 \@@_open_shorten:
4396 \int_if_zero:nTF { #1 }
4397 { \color { nicematrix-first-row } }
4398 {

```

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.

```

4399 \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4400 { \color { nicematrix-last-row } }
4401 }
4402 \keys_set:nn { nicematrix / xdots } { #3 }
4403 \@@_color:o \l_@@_xdots_color_tl
4404 \@@_actually_draw_Cdots:
4405 \group_end:
4406 }
4407 }

```

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.

```

4408 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4409 {
4410   \bool_if:NTF \l_@@_initial_open_bool
4411     { \@@_open_x_initial_dim: }
4412     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4413   \bool_if:NTF \l_@@_final_open_bool
4414     { \@@_open_x_final_dim: }
4415     { \@@_set_final_coords_from_anchor:n { mid-west } }
4416   \bool_lazy_and:nnTF
4417     { \l_@@_initial_open_bool }
4418     { \l_@@_final_open_bool }
4419     {
4420       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4421       \dim_set_eq:NN \l_tmpa_dim \pgf@y
4422       \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4423       \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4424       \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4425     }
4426   {
4427     \bool_if:NT \l_@@_initial_open_bool
4428       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4429     \bool_if:NT \l_@@_final_open_bool
4430       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4431   }
4432   \@@_draw_line:
4433 }
4434 \cs_new_protected:Npn \@@_open_y_initial_dim:
4435 {
4436   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4437   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4438   {
4439     \cs_if_exist:cT
4440       { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4441       {
4442         \pgfpointanchor
4443           { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4444           { north }
4445         \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4446         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4447       }
4448   }
4449   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4450   {
4451     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4452     \dim_set:Nn \l_@@_y_initial_dim
4453       {
4454         \fp_to_dim:n
4455           {
4456             \pgf@y
4457             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4458           }
4459       }
4460   }
4461 }

```

```

4462 \cs_new_protected:Npn \@@_open_y_final_dim:
4463 {
4464   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4465   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4466   {
4467     \cs_if_exist:cT
4468     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4469     {
4470       \pgfpointanchor
4471       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4472       { south }
4473       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4474       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4475     }
4476   }
4477   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4478   {
4479     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4480     \dim_set:Nn \l_@@_y_final_dim
4481     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4482   }
4483 }

```

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.

```

4484 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4485 {
4486   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4487   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4488   {
4489     \@@_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.

```

4490   \group_begin:
4491   \@@_open_shorten:
4492   \int_if_zero:nTF { #2 }
4493   { \color { nicematrix-first-col } }
4494   {
4495     \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4496     { \color { nicematrix-last-col } }
4497   }
4498   \keys_set:nn { nicematrix / xdots } { #3 }
4499   \@@_color:o \l_@@_xdots_color_tl
4500   \bool_if:NTF \l_@@_Vbrace_bool
4501   { \@@_actually_draw_Vbrace: }
4502   { \@@_actually_draw_Vdots: }
4503   \group_end:
4504 }
4505 }

```

The following function is used by regular calls of `\Vdots` or `\Vdotsfor` but not by `\Vbrace`. The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

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



```

4506 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4507 {
4508   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4509   { \@@_actually_draw_Vdots_i: }
4510   { \@@_actually_draw_Vdots_ii: }
4511   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4512   \@@_draw_line:
4513 }

```

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

```

4514 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4515 {
4516   \@@_open_y_initial_dim:
4517   \@@_open_y_final_dim:
4518   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4519 {
4520   \@@_qpoint:n { col - 1 }
4521   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4522   \dim_sub:Nn \l_@@_x_initial_dim
4523   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4524 }
4525 {
4526   \bool_lazy_and:nnTF
4527   { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4528   {
4529     \int_compare_p:nNn
4530     { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4531   }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4532 {
4533   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4534   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4535   \dim_add:Nn \l_@@_x_initial_dim
4536   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4537 }

```

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

```

4538 {
4539   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4540   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4541   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4542   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4543 }
4544 }
4545 }

```

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

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

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

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

```

4546 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4547 {
4548   \bool_set_false:N \l_tmpa_bool
4549   \bool_if:NF \l_@@_initial_open_bool
4550   {
4551     \bool_if:NF \l_@@_final_open_bool
4552     {
4553       \@@_set_initial_coords_from_anchor:n { south-west }
4554       \@@_set_final_coords_from_anchor:n { north-west }
4555       \bool_set:Nn \l_tmpa_bool

```

```

4556         {
4557             \dim_compare_p:nNn
4558             { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4559         }
4560     }
4561 }

```

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

```

4562 \bool_if:NTF \l_@@_initial_open_bool
4563 {
4564     \@@_open_y_initial_dim:
4565     \@@_set_final_coords_from_anchor:n { north }
4566     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4567 }
4568 {
4569     \@@_set_initial_coords_from_anchor:n { south }
4570     \bool_if:NTF \l_@@_final_open_bool
4571     { \@@_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.

```

4572     {
4573         \@@_set_final_coords_from_anchor:n { north }
4574         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4575         {
4576             \dim_set:Nn \l_@@_x_initial_dim
4577             {
4578                 \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4579                 \l_@@_x_initial_dim \l_@@_x_final_dim
4580             }
4581         }
4582     }
4583 }
4584 }

```

The following function is used by `\Vbrace` but not by regular uses of `\Vdots` or `\Vdotsfor`. The command `\@@_actually_draw_Vbrace:` has the following implicit arguments:

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

```

4585 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4586 {
4587     \bool_if:NTF \l_@@_initial_open_bool
4588     { \@@_open_y_initial_dim: }
4589     { \@@_set_initial_coords_from_anchor:n { south } }
4590     \bool_if:NTF \l_@@_final_open_bool
4591     { \@@_open_y_final_dim: }
4592     { \@@_set_final_coords_from_anchor:n { north } }

```

Now, we have the correct values for the  $y$ -values of both extremities of the brace. We have to compute the  $x$ -value (there is only one  $x$ -value since, of course, the brace is vertical).

If we are in the first (exterior) column, the brace must be drawn right flush.

```

4593 \int_if_zero:nTF { \l_@@_initial_j_int }
4594 {
4595     \@@_qpoint:n { col - 1 }

```

```

4596     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4597     \dim_sub:Nn \l_@@_x_initial_dim
4598     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4599 }

```

Elsewhere, the brace must be drawn left flush.

```

4600 {
4601   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4602   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4603   \dim_add:Nn \l_@@_x_initial_dim
4604   { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4605 }

```

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

```

4606   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4607   \@@_draw_line:
4608 }

```

```

4609 \cs_new:Npn \@@_colsep:
4610 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4611 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4612 {
4613   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4614   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4615   {
4616     \@@_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.

```

4617   \group_begin:
4618   \@@_open_shorten:
4619   \keys_set:nn { nicematrix / xdots } { #3 }
4620   \@@_color:o \l_@@_xdots_color_tl
4621   \@@_actually_draw_Ddots:
4622   \group_end:
4623 }
4624 }

```

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`.

```

4625 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4626 {
4627   \bool_if:NTF \l_@@_initial_open_bool
4628   {
4629     \@@_open_y_initial_dim:
4630     \@@_open_x_initial_dim:
4631   }
4632   { \@@_set_initial_coords_from_anchor:n { south-east } }
4633   \bool_if:NTF \l_@@_final_open_bool
4634   {
4635     \@@_open_x_final_dim:
4636     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4637   }
4638   { \@@_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.

```

4639   \bool_if:NT \l_@@_parallelize_diags_bool
4640   {
4641     \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).

```

4642     \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.

```

4643     {
4644       \dim_gset:Nn \g_@@_delta_x_one_dim
4645       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4646       \dim_gset:Nn \g_@@_delta_y_one_dim
4647       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4648     }

```

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`.

```

4649     {
4650       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4651       {
4652         \dim_set:Nn \l_@@_y_final_dim
4653         {
4654           \l_@@_y_initial_dim +
4655           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4656           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4657         }
4658       }
4659     }
4660   }
4661   \@@_draw_line:
4662 }

```

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.

```

4663 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4664 {
4665   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4666   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4667   {
4668     \@@_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.

```

4669   \group_begin:

```

```

4670     \@@_open_shorten:
4671     \keys_set:nn { nicematrix / xdots } { #3 }
4672     \@@_color:o \l_@@_xdots_color_tl
4673     \@@_actually_draw_Iddots:
4674   \group_end:
4675 }
4676 }

```

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.`

```

4677 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4678 {
4679   \bool_if:NTF \l_@@_initial_open_bool
4680   {
4681     \@@_open_y_initial_dim:
4682     \@@_open_x_initial_dim:
4683   }
4684   { \@@_set_initial_coords_from_anchor:n { south-west } }
4685   \bool_if:NTF \l_@@_final_open_bool
4686   {
4687     \@@_open_y_final_dim:
4688     \@@_open_x_final_dim:
4689   }
4690   { \@@_set_final_coords_from_anchor:n { north-east } }
4691   \bool_if:NT \l_@@_parallelize_diags_bool
4692   {
4693     \int_gincr:N \g_@@_iddots_int
4694     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4695     {
4696       \dim_gset:Nn \g_@@_delta_x_two_dim
4697       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4698       \dim_gset:Nn \g_@@_delta_y_two_dim
4699       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4700     }
4701     {
4702       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4703       {
4704         \dim_set:Nn \l_@@_y_final_dim
4705         {
4706           \l_@@_y_initial_dim +
4707           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4708           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4709         }
4710       }
4711     }
4712   }
4713   \@@_draw_line:
4714 }

```

## 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`

```

4715 \cs_new_protected:Npn \@@_draw_line:
4716 {
4717   \pgfrememberpicturepositiononpagetrue
4718   \pgf@relevantforpicturesizefalse
4719   \bool_lazy_or:nnTF
4720     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4721     { \l_@@_dotted_bool }
4722     { \@@_draw_standard_dotted_line: }
4723     { \@@_draw_unstandard_dotted_line: }
4724 }
```

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.

```

4725 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4726 {
4727   \begin { scope }
4728     \@@_draw_unstandard_dotted_line:o
4729     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4730 }
```

We have used the fact that, in PGF, an 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.

```

4731 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4732 {
4733   \@@_draw_unstandard_dotted_line:nooo
4734   { #1 }
4735   \l_@@_xdots_up_tl
4736   \l_@@_xdots_down_tl
4737   \l_@@_xdots_middle_tl
4738 }
4739 \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.

```

4740 \hook_gput_code:nnn { begindocument } { . }
4741 {
4742   \IfPackageLoadedT { tikz }
4743   {
4744     \tikzset
4745     {
4746       @@_node_above / .style = { sloped , above } ,
4747       @@_node_below / .style = { sloped , below } ,
4748       @@_node_middle / .style =
4749       {
```

```

4750         sloped ,
4751         inner~sep = \c_@@_innersep_middle_dim
4752     }
4753 }
4754 }
4755 }

```

```

4756 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4757 {

```

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.

```

4758     \dim_zero_new:N \l_@@_l_dim
4759     \dim_set:Nn \l_@@_l_dim
4760     {
4761         \fp_to_dim:n
4762         {
4763             sqrt
4764             (
4765                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4766                 +
4767                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4768             )
4769         }
4770     }

```

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.

```

4771     \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4772     {
4773         \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4774         \@@_draw_unstandard_dotted_line_i:
4775     }

```

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

```

4776     \bool_if:NT \l_@@_xdots_h_labels_bool
4777     {
4778         \tikzset
4779         {
4780             @@_node_above / .style = { auto = left } ,
4781             @@_node_below / .style = { auto = right } ,
4782             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4783         }
4784     }
4785     \tl_if_empty:nF { #4 }
4786     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4787     \draw
4788     [ #1 ]
4789     ( \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).

```

4790     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4791     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4792     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4793     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4794     \end { scope }
4795 }
4796 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

```

4797 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4798 {
4799   \dim_set:Nn \l_tmpa_dim
4800   {
4801     \l_@@_x_initial_dim
4802     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4803     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4804   }
4805   \dim_set:Nn \l_tmpb_dim
4806   {
4807     \l_@@_y_initial_dim
4808     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4809     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4810   }
4811   \dim_set:Nn \l_@@_tmpc_dim
4812   {
4813     \l_@@_x_final_dim
4814     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4815     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4816   }
4817   \dim_set:Nn \l_@@_tmpd_dim
4818   {
4819     \l_@@_y_final_dim
4820     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4821     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4822   }
4823   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4824   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4825   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4826   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4827 }

```

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

```

4828 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4829 {
4830   \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.

```

4831   \dim_zero_new:N \l_@@_l_dim
4832   \dim_set:Nn \l_@@_l_dim
4833   {
4834     \fp_to_dim:n
4835     {
4836       sqrt
4837       (
4838         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4839         +
4840         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4841       )
4842     }
4843   }

```

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.

```

4844   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4845   {
4846     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4847     { \@@_draw_standard_dotted_line_i: }
4848   }
4849   \group_end:

```



```

4850 \bool_lazy_all:nF
4851 {
4852   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4853   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4854   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4855 }
4856 { \@@_labels_standard_dotted_line: }
4857 }
4858 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4859 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4860 {

```

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

```

4861 \int_set:Nn \l_tmpa_int
4862 {
4863   \dim_ratio:nn
4864   {
4865     \l_@@_l_dim
4866     - \l_@@_xdots_shorten_start_dim
4867     - \l_@@_xdots_shorten_end_dim
4868   }
4869   { \l_@@_xdots_inter_dim }
4870 }

```

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

```

4871 \dim_set:Nn \l_tmpa_dim
4872 {
4873   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4874   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4875 }
4876 \dim_set:Nn \l_tmpb_dim
4877 {
4878   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4879   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4880 }

```

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.

```

4881 \dim_gadd:Nn \l_@@_x_initial_dim
4882 {
4883   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4884   \dim_ratio:nn
4885   {
4886     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4887     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4888   }
4889   { 2 \l_@@_l_dim }
4890 }
4891 \dim_gadd:Nn \l_@@_y_initial_dim
4892 {
4893   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4894   \dim_ratio:nn
4895   {
4896     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4897     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4898   }
4899   { 2 \l_@@_l_dim }
4900 }
4901 \pgf@relevantforpicturesizefalse
4902 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4903 {
4904   \pgfpathcircle

```

```

4905         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4906         { \l_@@_xdots_radius_dim }
4907         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4908         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4909     }
4910     \pgfusepathqfill
4911 }

4912 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4913 {
4914     \pgfscope
4915     \pgftransformshift
4916     {
4917         \pgfpointlineattime { 0.5 }
4918         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4919         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4920     }
4921     \fp_set:Nn \l_tmpa_fp
4922     {
4923         atand
4924         (
4925             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4926             \l_@@_x_final_dim - \l_@@_x_initial_dim
4927         )
4928     }
4929     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4930     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4931     \tl_if_empty:NF \l_@@_xdots_middle_tl
4932     {
4933         \begin { pgfscope }
4934         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4935         \pgfnode
4936         { rectangle }
4937         { center }
4938         {
4939             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4940             {
4941                 \c_math_toggle_token
4942                 \scriptstyle \l_@@_xdots_middle_tl
4943                 \c_math_toggle_token
4944             }
4945         }
4946         { }
4947         {
4948             \pgfsetfillcolor { white }
4949             \pgfusepath { fill }
4950         }
4951         \end { pgfscope }
4952     }
4953     \tl_if_empty:NF \l_@@_xdots_up_tl
4954     {
4955         \pgfnode
4956         { rectangle }
4957         { south }
4958         {
4959             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4960             {
4961                 \c_math_toggle_token
4962                 \scriptstyle \l_@@_xdots_up_tl
4963                 \c_math_toggle_token
4964             }
4965         }
4966         { }

```

```

4967         { \pgfusepath { } }
4968     }
4969     \tl_if_empty:NF \l_@@_xdots_down_tl
4970     {
4971         \pgfnode
4972         { rectangle }
4973         { north }
4974         {
4975             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4976             {
4977                 \c_math_toggle_token
4978                 \scriptstyle \l_@@_xdots_down_tl
4979                 \c_math_toggle_token
4980             }
4981         }
4982         { }
4983         { \pgfusepath { } }
4984     }
4985     \endpgfscope
4986 }

```

## 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.

```

4987 \hook_gput_code:nnn { begindocument } { . }
4988 {

```

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

```

4989     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
4990     \cs_new_protected:Npn \@@_Ldots:
4991     { \@@_collect_options:n { \@@_Ldots_i } }
4992     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4993     {
4994         \int_if_zero:nTF { \c_jCol }
4995         { \@@_error:nn { in~first~col } { \Ldots } }
4996         {
4997             \int_compare:nNnTF { \c_jCol } = { \l_@@_last_col_int }
4998             { \@@_error:nn { in~last~col } { \Ldots } }
4999             {
5000                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5001                 { #1 , down = #2 , up = #3 , middle = #4 }
5002             }
5003         }
5004         \bool_if:NF \l_@@_nullify_dots_bool
5005         { \phantom { \ensuremath { \@@_old_ldots: } } }
5006         \bool_gset_true:N \g_@@_empty_cell_bool
5007     }

5008     \cs_new_protected:Npn \@@_Cdots:

```

```

5009 { \@@_collect_options:n { \@@_Cdots_i } }
5010 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5011 {
5012   \int_if_zero:nTF { \c@jCol }
5013   { \@@_error:nn { in~first~col } { \Cdots } }
5014   {
5015     \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5016     { \@@_error:nn { in~last~col } { \Cdots } }
5017     {
5018       \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5019       { #1 , down = #2 , up = #3 , middle = #4 }
5020     }
5021   }
5022   \bool_if:NF \l_@@_nullify_dots_bool
5023   { \phantom { \ensuremath { \@@_old_cdots: } } }
5024   \bool_gset_true:N \g_@@_empty_cell_bool
5025 }

5026 \cs_new_protected:Npn \@@_Vdots:
5027 { \@@_collect_options:n { \@@_Vdots_i } }
5028 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5029 {
5030   \int_if_zero:nTF { \c@iRow }
5031   { \@@_error:nn { in~first~row } { \Vdots } }
5032   {
5033     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5034     { \@@_error:nn { in~last~row } { \Vdots } }
5035     {
5036       \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5037       { #1 , down = #2 , up = #3 , middle = #4 }
5038     }
5039   }
5040   \bool_if:NF \l_@@_nullify_dots_bool
5041   { \phantom { \ensuremath { \@@_old_vdots: } } }
5042   \bool_gset_true:N \g_@@_empty_cell_bool
5043 }

5044 \cs_new_protected:Npn \@@_Ddots:
5045 { \@@_collect_options:n { \@@_Ddots_i } }
5046 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5047 {
5048   \int_case:nnF \c@iRow
5049   {
5050     0 { \@@_error:nn { in~first~row } { \Ddots } }
5051     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5052   }
5053   {
5054     \int_case:nnF \c@jCol
5055     {
5056       0 { \@@_error:nn { in~first~col } { \Ddots } }
5057       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5058     }
5059     {
5060       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5061       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5062       { #1 , down = #2 , up = #3 , middle = #4 }
5063     }
5064   }
5065 }
5066 \bool_if:NF \l_@@_nullify_dots_bool
5067 { \phantom { \ensuremath { \@@_old_ddots: } } }
5068 \bool_gset_true:N \g_@@_empty_cell_bool
5069 }

```

```

5070 \cs_new_protected:Npn \@@_Iddots:
5071 { \@@_collect_options:n { \@@_Iddots_i } }
5072 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5073 {
5074   \int_case:nnF \c@iRow
5075   {
5076     0 { \@@_error:nn { in~first~row } { \Iddots } }
5077     \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5078   }
5079   {
5080     \int_case:nnF \c@jCol
5081     {
5082       0 { \@@_error:nn { in~first~col } { \Iddots } }
5083       \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5084     }
5085     {
5086       \keys_set_known:nn { nicematrix / Ddots } { #1 }
5087       \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5088       { #1 , down = #2 , up = #3 , middle = #4 }
5089     }
5090   }
5091   \bool_if:NF \l_@@_nullify_dots_bool
5092   { \phantom { \ensuremath { \@@_old_iddots: } } }
5093   \bool_gset_true:N \g_@@_empty_cell_bool
5094 }
5095 }

```

End of the \AddToHook.

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

```

5096 \keys_define:nn { nicematrix / Ddots }
5097 {
5098   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5099   draw-first .default:n = true ,
5100   draw-first .value_forbidden:n = true
5101 }

```

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

```

5102 \cs_new_protected:Npn \@@_Hspace:
5103 {
5104   \bool_gset_true:N \g_@@_empty_cell_bool
5105   \hspace
5106 }

```

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.

```

5107 \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.

```

5108 \cs_new:Npn \@@_Hdotsfor:
5109 {
5110   \bool_lazy_and:nnTF
5111   { \int_if_zero_p:n { \c@jCol } }
5112   { \int_if_zero_p:n { \l_@@_first_col_int } }
5113   {
5114     \bool_if:NTF \g_@@_after_col_zero_bool
5115     {
5116       \multicolumn { 1 } { c } { }
5117       \@@_Hdotsfor_i:

```

```

5118     }
5119     { \@@_fatal:n { Hdotsfor~in~col-0 } }
5120 }
5121 {
5122     \multicolumn { 1 } { c } { }
5123     \@@_Hdotsfor_i:
5124 }
5125 }

```

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:`).

```

5126 \hook_gput_code:nnn { begindocument } { . }
5127 {

```

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

```

5128     \cs_new_protected:Npn \@@_Hdotsfor_i:
5129     { \@@_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`).

```

5130     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5131     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5132     {
5133         \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5134         {
5135             \@@_Hdotsfor:nnnn
5136             { \int_use:N \c@iRow }
5137             { \int_use:N \c@jCol }
5138             { #2 }
5139             {
5140                 #1 , #3 ,
5141                 down = \exp_not:n { #4 } ,
5142                 up = \exp_not:n { #5 } ,
5143                 middle = \exp_not:n { #6 }
5144             }
5145         }
5146         \prg_replicate:nn { #2 - 1 }
5147         {
5148             &
5149             \multicolumn { 1 } { c } { }
5150             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5151         }
5152     }
5153 }

```

```

5154 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5155 {
5156     \bool_set_false:N \l_@@_initial_open_bool
5157     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5158     \int_set:Nn \l_@@_initial_i_int { #1 }
5159     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5160     \int_compare:nNnTF { #2 } = { \c_one_int }
5161     {
5162         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5163         \bool_set_true:N \l_@@_initial_open_bool
5164     }
5165     {
5166         \cs_if_exist:cTF
5167         {

```

```

5168         pgf @ sh @ ns @ \@@_env:
5169         - \int_use:N \l_@@_initial_i_int
5170         - \int_eval:n { #2 - 1 }
5171     }
5172     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5173     {
5174         \int_set:Nn \l_@@_initial_j_int { #2 }
5175         \bool_set_true:N \l_@@_initial_open_bool
5176     }
5177 }
5178 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5179 {
5180     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5181     \bool_set_true:N \l_@@_final_open_bool
5182 }
5183 {
5184     \cs_if_exist:cTF
5185     {
5186         pgf @ sh @ ns @ \@@_env:
5187         - \int_use:N \l_@@_final_i_int
5188         - \int_eval:n { #2 + #3 }
5189     }
5190     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5191     {
5192         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5193         \bool_set_true:N \l_@@_final_open_bool
5194     }
5195 }
5196 \group_begin:
5197 \@@_open_shorten:
5198 \int_if_zero:nTF { #1 }
5199 { \color { nicematrix-first-row } }
5200 {
5201     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5202     { \color { nicematrix-last-row } }
5203 }
5204 \keys_set:nn { nicematrix / xdots } { #4 }
5205 \@@_color:o \l_@@_xdots_color_tl
5206 \@@_actually_draw_ldots:
5207 \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).

```

5208     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5209     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5210 }

```

```

5211 \hook_gput_code:nnn { begindocument } { . }
5212 {
5213     \cs_new_protected:Npn \@@_Vdotsfor:
5214     { \@@_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`).

```

5215     \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5216     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5217     {
5218         \bool_gset_true:N \g_@@_empty_cell_bool
5219         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5220         {
5221             \@@_Vdotsfor:nnnn

```

```

5222         { \int_use:N \c@iRow }
5223         { \int_use:N \c@jCol }
5224         { #2 }
5225         {
5226             #1 , #3 ,
5227             down = \exp_not:n { #4 } ,
5228             up = \exp_not:n { #5 } ,
5229             middle = \exp_not:n { #6 }
5230         }
5231     }
5232 }
5233 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5234 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5235 {
5236     \bool_set_false:N \l_@@_initial_open_bool
5237     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5238     \int_set:Nn \l_@@_initial_j_int { #2 }
5239     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5240     \int_compare:nNnTF { #1 } = { \c_one_int }
5241     {
5242         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5243         \bool_set_true:N \l_@@_initial_open_bool
5244     }
5245     {
5246         \cs_if_exist:cTF
5247         {
5248             pgf @ sh @ ns @ \@@_env:
5249             - \int_eval:n { #1 - 1 }
5250             - \int_use:N \l_@@_initial_j_int
5251         }
5252         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5253         {
5254             \int_set:Nn \l_@@_initial_i_int { #1 }
5255             \bool_set_true:N \l_@@_initial_open_bool
5256         }
5257     }
5258     \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5259     {
5260         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5261         \bool_set_true:N \l_@@_final_open_bool
5262     }
5263     {
5264         \cs_if_exist:cTF
5265         {
5266             pgf @ sh @ ns @ \@@_env:
5267             - \int_eval:n { #1 + #3 }
5268             - \int_use:N \l_@@_final_j_int
5269         }
5270         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5271         {
5272             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5273             \bool_set_true:N \l_@@_final_open_bool
5274         }
5275     }

```



```

5276 \group_begin:
5277 \@@_open_shorten:
5278 \int_if_zero:nTF { #2 }
5279 { \color { nicematrix-first-col } }
5280 {
5281   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5282   { \color { nicematrix-last-col } }
5283 }
5284 \keys_set:nn { nicematrix / xdots } { #4 }
5285 \@@_color:o \l_@@_xdots_color_tl
5286 \bool_if:NTF \l_@@_Vbrace_bool
5287 { \@@_actually_draw_Vbrace: }
5288 { \@@_actually_draw_Vdots: }
5289 \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).

```

5290 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5291 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5292 }

```

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

```

5293 \NewDocumentCommand \@@_rotate: { 0 { } }
5294 {
5295   \bool_gset_true:N \g_@@_rotate_bool
5296   \keys_set:nn { nicematrix / rotate } { #1 }
5297   \ignorespaces
5298 }
5299 \keys_define:nn { nicematrix / rotate }
5300 {
5301   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5302   c .value_forbidden:n = true ,
5303   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5304 }

```

## 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>

```

5305 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop

```

---

<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`.

```

5306 {
5307   \tl_if_empty:nTF { #2 }
5308     { #1 }
5309     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5310 }
5311 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5312 { \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).

```

5313 \hook_gput_code:nnn { begindocument } { . }
5314 {

```

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

```

5315   \tl_set_rescan:Nnn \l_tmpa_tl { }
5316   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5317   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5318     {
5319     \group_begin:
5320     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5321     \@@_color:o \l_@@_xdots_color_tl
5322     \use:e
5323     {
5324       \@@_line_i:nn
5325       { \@@_double_int_eval:n #2 - \q_stop }
5326       { \@@_double_int_eval:n #3 - \q_stop }
5327     }
5328     \group_end:
5329   }
5330 }

5331 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5332 {
5333   \bool_set_false:N \l_@@_initial_open_bool
5334   \bool_set_false:N \l_@@_final_open_bool
5335   \bool_lazy_or:nnTF
5336     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5337     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5338     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5339   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5340 }

5341 \hook_gput_code:nnn { begindocument } { . }
5342 {
5343   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5344   {

```

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:`.

```

5345     \c_@@_pgfortikzpicture_tl
5346     \@@_draw_line_iii:nn { #1 } { #2 }
5347     \c_@@_endpgfortikzpicture_tl
5348   }
5349 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5350 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5351 {
5352   \pgfrememberpicturepositiononpagetrue

```

```

5353 \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5354 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5355 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5356 \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5357 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5358 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5359 \@@_draw_line:
5360 }

```

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.

```

5361 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5362 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }

```

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

```

5363 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5364 {
5365   \tl_gput_right:Ne \g_@@_row_style_tl
5366   {

```

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

```

5367 \exp_not:N
5368 \@@_if_row_less_than:nn
5369 { \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`).

```

5370 {
5371   \exp_not:N
5372   \@@_if_col_greater_than:nn
5373   { \int_eval:n { \c@jCol } }
5374   { \exp_not:n { #1 } \scan_stop: }
5375 }
5376 }
5377 }
5378 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5379 \keys_define:nn { nicematrix / RowStyle }
5380 {
5381   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5382   cell-space-top-limit .value_required:n = true ,
5383   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5384   cell-space-bottom-limit .value_required:n = true ,

```

```

5385 cell-space-limits .meta:n =
5386 {
5387     cell-space-top-limit = #1 ,
5388     cell-space-bottom-limit = #1 ,
5389 } ,
5390 color .tl_set:N = \l_@@_color_tl ,
5391 color .value_required:n = true ,
5392 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5393 bold .default:n = true ,
5394 nb-rows .code:n =
5395     \str_if_eq:eeTF { #1 } { * }
5396     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5397     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5398 nb-rows .value_required:n = true ,
5399 fill .tl_set:N = \l_@@_fill_tl ,
5400 fill .value_required:n = true ,
5401 opacity .tl_set:N = \l_@@_opacity_tl ,
5402 opacity .value_required:n = true ,
5403 rowcolor .tl_set:N = \l_@@_fill_tl ,
5404 rowcolor .value_required:n = true ,
5405 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5406 rounded-corners .default:n = 4 pt ,
5407 unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5408 }

```

```

5409 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5410 {
5411     \group_begin:
5412     \tl_clear:N \l_@@_fill_tl
5413     \tl_clear:N \l_@@_opacity_tl
5414     \tl_clear:N \l_@@_color_tl
5415     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5416     \dim_zero:N \l_@@_rounded_corners_dim
5417     \dim_zero:N \l_tmpa_dim
5418     \dim_zero:N \l_tmpb_dim
5419     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5420     \tl_if_empty:NF \l_@@_fill_tl
5421     {
5422         \@@_add_opacity_to_fill:
5423         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5424         {

```

The command \@@\_exp\_color\_arg:No is *fully expandable*.

```

5425         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5426         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5427         {
5428             \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5429             - *
5430         }
5431         { \dim_use:N \l_@@_rounded_corners_dim }
5432     }
5433 }
5434 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l\_tmpa\_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5435     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5436     {
5437         \@@_put_in_row_style:e
5438         {
5439             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5440             {

```

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

```

5441         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5442         { \dim_use:N \l_tmpa_dim }
5443     }
5444 }
5445 }

```

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

```

5446 \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5447 {
5448     \@@_put_in_row_style:e
5449     {
5450         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5451         {
5452             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5453             { \dim_use:N \l_tmpb_dim }
5454         }
5455     }
5456 }

```

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

```

5457 \tl_if_empty:NF \l_@@_color_tl
5458 {
5459     \@@_put_in_row_style:e
5460     {
5461         \mode_leave_vertical:
5462         \@@_color:n { \l_@@_color_tl }
5463     }
5464 }

```

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

```

5465 \bool_if:NT \l_@@_bold_row_style_bool
5466 {
5467     \@@_put_in_row_style:n
5468     {
5469         \exp_not:n
5470         {
5471             \if_mode_math:
5472                 \c_math_toggle_token
5473                 \bfseries \boldmath
5474                 \c_math_toggle_token
5475             \else:
5476                 \bfseries \boldmath
5477             \fi:
5478         }
5479     }
5480 }
5481 \group_end:
5482 \g_@@_row_style_tl
5483 \ignorespaces
5484 }

```

The following commande must *not* be protected.

```

5485 \cs_new:Npn \@@_rounded_from_row:n #1
5486 {
5487     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5488     { \int_eval:n { #1 } - 1 }
5489     {
5490         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5491         - \exp_not:n { \int_use:N \c@jCol }
5492     }
5493     { \dim_use:N \l_@@_rounded_corners_dim }
5494 }

```

## 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).

```
5495 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5496 {
```

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.

```
5497 \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`.

```
5498 \str_if_in:nnF { #1 } { !! }
5499 {
5500 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

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

```
5501 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5502 }
5503 \int_if_zero:nTF { \l_tmpa_int }
```

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

```
5504 {
5505 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5506 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5507 }
```

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

```
5508 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5509 }
5510 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5511 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5512 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5513 {
5514 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5515 {
```

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

```

5516     \group_begin:
5517     \pgfsetcornersarced
5518     {
5519         \pgfpoint
5520         { \l_@@_tab_rounded_corners_dim }
5521         { \l_@@_tab_rounded_corners_dim }
5522     }

```

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.

```

5523     \bool_if:NTF \l_@@_hvlines_bool
5524     {
5525         \pgfpathrectanglecorners
5526         {
5527             \pgfpointadd
5528             { \@@_qpoint:n { row-1 } }
5529             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5530         }
5531         {
5532             \pgfpointadd
5533             {
5534                 \@@_qpoint:n
5535                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5536             }
5537             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5538         }
5539     }
5540     {
5541         \pgfpathrectanglecorners
5542         { \@@_qpoint:n { row-1 } }
5543         {
5544             \pgfpointadd
5545             {
5546                 \@@_qpoint:n
5547                 { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5548             }
5549             { \pgfpoint \c_zero_dim \arrayrulewidth }
5550         }
5551     }
5552     \pgfusepath { clip }
5553     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5554     }
5555 }

```

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`).

```

5556 \cs_new_protected:Npn \@@_actually_color:
5557 {
5558     \pgfpicture
5559     \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.

```

5560     \@@_clip_with_rounded_corners:
5561     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5562     {
5563         \int_compare:nNnTF { ##1 } = { \c_one_int }

```

```

5564     {
5565         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5566         \use:c { g_@@_color _ 1 _tl }
5567         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5568     }
5569     {
5570         \begin { pgfscope }
5571             \@@_color_opacity: ##2
5572             \use:c { g_@@_color _ ##1 _tl }
5573             \tl_gclear:c { g_@@_color _ ##1 _tl }
5574             \pgfusepath { fill }
5575         \end { pgfscope }
5576     }
5577 }
5578 \endpgfpicture
5579 }

```

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

```

5580 \cs_new_protected:Npn \@@_color_opacity:
5581 {
5582     \peek_meaning:NTF [
5583         { \@@_color_opacity:w }
5584         { \@@_color_opacity:w [ ] }
5585     }

```

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 currying.

```

5586 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5587 {
5588     \tl_clear:N \l_tmpa_tl
5589     \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.

```

5590     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5591     \tl_if_empty:NTF \l_tmpb_tl
5592         { \@declaredcolor }
5593         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5594 }

```

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

```

5595 \keys_define:nn { nicematrix / color-opacity }
5596 {
5597     opacity .tl_set:N          = \l_tmpa_tl ,
5598     opacity .value_required:n = true
5599 }

```

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

```

5600 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5601 {
5602     \def \l_@@_rows_tl { #1 }
5603     \def \l_@@_cols_tl { #2 }
5604     \@@_cartesian_path:
5605 }

```

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

```

5606 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5607 {
5608     \tl_if_blank:nF { #2 }

```



```

5609 {
5610   \@@_add_to_colors_seq:en
5611   { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5612   { \@@_cartesian_color:nn { #3 } { - } }
5613 }
5614 }

```

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

```

5615 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5616 {
5617   \tl_if_blank:nF { #2 }
5618   {
5619     \@@_add_to_colors_seq:en
5620     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5621     { \@@_cartesian_color:nn { - } { #3 } }
5622   }
5623 }

```

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

```

5624 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5625 {
5626   \tl_if_blank:nF { #2 }
5627   {
5628     \@@_add_to_colors_seq:en
5629     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5630     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5631   }
5632 }

```

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

```

5633 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5634 {
5635   \tl_if_blank:nF { #2 }
5636   {
5637     \@@_add_to_colors_seq:en
5638     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5639     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5640   }
5641 }

```

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

```

5642 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5643 {
5644   \@@_cut_on_hyphen:w #1 \q_stop
5645   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5646   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5647   \@@_cut_on_hyphen:w #2 \q_stop
5648   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5649   \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`.

```

5650   \@@_cartesian_path:n { #3 }
5651 }

```

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

```

5652 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5653 {
5654   \clist_map_inline:nn { #3 }
5655   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5656 }

```

```

5657 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5658 {
5659   \int_step_inline:nn { \c@iRow }
5660   {
5661     \int_step_inline:nn { \c@jCol }
5662     {
5663       \int_if_even:nTF { #####1 + ##1 }
5664       { \@@_cellcolor [ #1 ] { #2 } }
5665       { \@@_cellcolor [ #1 ] { #3 } }
5666       { ##1 - #####1 }
5667     }
5668   }
5669 }

```

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”.

```

5670 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5671 {
5672   \@@_rectanglecolor [ #1 ] { #2 }
5673   { 1 - 1 }
5674   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5675 }

```

```

5676 \keys_define:nn { nicematrix / rowcolors }
5677 {
5678   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5679   respect-blocks .default:n = true ,
5680   cols .tl_set:N = \l_@@_cols_tl ,
5681   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5682   restart .default:n = true ,
5683   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5684 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

**#1** (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```

5685 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5686 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5687   \group_begin:
5688   \seq_clear_new:N \l_@@_colors_seq
5689   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5690   \tl_clear_new:N \l_@@_cols_tl
5691   \tl_set:Nn \l_@@_cols_tl { - }
5692   \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).

```

5693   \int_zero_new:N \l_@@_color_int
5694   \int_set_eq:NN \l_@@_color_int \c_one_int
5695   \bool_if:NT \l_@@_respect_blocks_bool
5696   {

```

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`).

```

5697     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5698     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5699     { \@@_not_in_exterior_p:nnnnn ##1 }
5700   }
5701   \pgfpicture
5702   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5703   \clist_map_inline:nn { #2 }
5704   {
5705     \tl_set:Nn \l_tmpa_tl { ##1 }
5706     \tl_if_in:NnTF \l_tmpa_tl { - }
5707     { \@@_cut_on_hyphen:w ##1 \q_stop }
5708     { \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.

```

5709     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5710     \int_set:Nn \l_@@_color_int
5711     { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5712     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5713     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5714     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5715         \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).

```

5716         \bool_if:NT \l_@@_respect_blocks_bool
5717         {
5718           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5719           { \@@_intersect_our_row_p:nnnnn #####1 }
5720           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5721         }
5722         \tl_set:Nn \l_@@_rows_tl
5723         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5724         \tl_set:Nn \l_@@_color_tl
5725         {
5726           \@@_color_index:n
5727           {
5728             \int_mod:nn
5729             { \l_@@_color_int - 1 }
5730             { \seq_count:N \l_@@_colors_seq }
5731             + 1
5732           }
5733         }
5734         \tl_if_empty:NF \l_@@_color_tl
5735         {
5736           \@@_add_to_colors_seq:ee
5737           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5738           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5739         }
5740         \int_incr:N \l_@@_color_int
5741         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5742       }
5743     }
5744   \endpgfpicture
5745   \group_end:
5746 }

```

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.

```
5747 \cs_new:Npn \@@_color_index:n #1
5748 {
```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```
5749 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5750 { \@@_color_index:n { #1 - 1 } }
5751 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5752 }
```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```
5753 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5754 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5755 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5756 {
5757   \int_compare:nNtT { #3 } > { \l_tmpb_int }
5758   { \int_set:Nn \l_tmpb_int { #3 } }
5759 }
```

```
5760 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5761 {
5762   \int_if_zero:nTF { #4 }
5763   { \prg_return_false: }
5764   {
5765     \int_compare:nNtTF { #2 } > { \c@jCol }
5766     { \prg_return_false: }
5767     { \prg_return_true: }
5768   }
5769 }
```

The following command return true when the block intersects the row `\l_tmpa_int`.

```
5770 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5771 {
5772   \int_compare:nNtTF { #1 } > { \l_tmpa_int }
5773   { \prg_return_false: }
5774   {
5775     \int_compare:nNtTF { \l_tmpa_int } > { #3 }
5776     { \prg_return_false: }
5777     { \prg_return_true: }
5778   }
5779 }
```

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`).

```
5780 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5781 {
5782   \dim_compare:nNtTF { #1 } = { \c_zero_dim }
5783   {
5784     \bool_if:NtF \l_@@_nocolor_used_bool
5785     { \@@_cartesian_path_normal_ii: }
```

```

5786     {
5787         \clist_if_empty:NTF \l_@@_corners_cells_clist
5788         { \@@_cartesian_path_normal_i:n { #1 } }
5789         { \@@_cartesian_path_normal_ii: }
5790     }
5791 }
5792 { \@@_cartesian_path_normal_i:n { #1 } }
5793 }

```

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.

```

5794 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5795 {
5796     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5797     \clist_map_inline:Nn \l_@@_cols_tl
5798     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5799         \def \l_tmpa_tl { ##1 }
5800         \tl_if_in:NnTF \l_tmpa_tl { - }
5801         { \@@_cut_on_hyphen:w ##1 \q_stop }
5802         { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5803         \tl_if_empty:NTF \l_tmpa_tl
5804         { \def \l_tmpa_tl { 1 } }
5805         {
5806             \str_if_eq:eeT { \l_tmpa_tl } { * }
5807             { \def \l_tmpa_tl { 1 } }
5808         }
5809         \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5810         { \@@_error:n { Invalid~col~number } }
5811         \tl_if_empty:NTF \l_tmpb_tl
5812         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5813         {
5814             \str_if_eq:eeT { \l_tmpb_tl } { * }
5815             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5816         }
5817         \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5818         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5819         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5820         \@@_qpoint:n { col - \l_tmpa_tl }
5821         \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5822         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5823         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5824         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5825         \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.

```

5826     \clist_map_inline:Nn \l_@@_rows_tl
5827     {
5828         \def \l_tmpa_tl { #####1 }
5829         \tl_if_in:NnTF \l_tmpa_tl { - }
5830         { \@@_cut_on_hyphen:w #####1 \q_stop }
5831         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5832         \tl_if_empty:NTF \l_tmpa_tl
5833         { \def \l_tmpa_tl { 1 } }
5834         {
5835             \str_if_eq:eeT { \l_tmpa_tl } { * }
5836             { \def \l_tmpa_tl { 1 } }
5837         }
5838         \tl_if_empty:NTF \l_tmpb_tl

```

```

5839         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5840     {
5841         \str_if_eq:eeT { \l_tmpb_tl } { * }
5842         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5843     }
5844     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5845     { \@@_error:n { Invalid-row-number } }
5846     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5847     { \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`.

```

5848     \cs_if_exist:cF
5849     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5850     {
5851         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5852         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5853         \@@_qpoint:n { row - \l_tmpa_tl }
5854         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5855         \pgfpathrectanglecorners
5856         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5857         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5858     }
5859 }
5860 }
5861 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5862 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5863 {
5864     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5865     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5866     \clist_map_inline:Nn \l_@@_cols_tl
5867     {
5868         \@@_qpoint:n { col - ##1 }
5869         \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5870         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5871         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5872         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5873         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5874     \clist_map_inline:Nn \l_@@_rows_tl
5875     {
5876         \@@_if_in_corner:nF { #####1 - ##1 }
5877         {
5878             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5879             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5880             \@@_qpoint:n { row - #####1 }
5881             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5882             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5883             {
5884                 \pgfpathrectanglecorners
5885                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5886                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5887             }
5888         }
5889     }
5890 }
5891 }

```

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`).

```
5892 \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

```
5893 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5894 {
5895   \bool_set_true:N \l_@@_nocolor_used_bool
5896   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5897   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5898   \clist_map_inline:Nn \l_@@_rows_tl
5899   {
5900     \clist_map_inline:Nn \l_@@_cols_tl
5901     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5902   }
5903 }
```

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`.

```
5904 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5905 {
5906   \clist_set_eq:NN \l_tmpa_clist #1
5907   \clist_clear:N #1
5908   \clist_map_inline:Nn \l_tmpa_clist
5909   {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
5910     \def \l_tmpa_tl { ##1 }
5911     \tl_if_in:NnTF \l_tmpa_tl { - }
5912     { \@@_cut_on_hyphen:w ##1 \q_stop }
5913     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5914     \bool_lazy_or:nnT
5915     { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
5916     { \tl_if_blank_p:o \l_tmpa_tl }
5917     { \def \l_tmpa_tl { 1 } }
5918     \bool_lazy_or:nnT
5919     { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
5920     { \tl_if_blank_p:o \l_tmpb_tl }
5921     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5922     \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5923     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5924     \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5925     { \clist_put_right:Nn #1 { ####1 } }
5926   }
5927 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5928 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5929 {
5930   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5931   {
```

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`).

```
5932     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5933     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5934   }
5935   \ignorespaces
5936 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

5937 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5938 {
5939   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5940   {
5941     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5942     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5943     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5944   }
5945   \ignorespaces
5946 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5947 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5948 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5949 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5950 {

```

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`.

```

5951   \seq_gclear:N \g_tmpa_seq
5952   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5953     { \@@_rowlistcolors_tabular:nnnn #1 }
5954   \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`.

```

5955   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5956   {
5957     { \int_use:N \c@iRow }
5958     { \exp_not:n { #1 } }
5959     { \exp_not:n { #2 } }
5960     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5961   }
5962   \ignorespaces
5963 }

```

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`).

```

5964 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5965 {
5966   \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).

```

5967   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5968   {
5969     \tl_gput_right:Ne \g_@@_pre_code_before_tl

```



```

5970     {
5971         \@@_rowlistcolors
5972         [ \exp_not:n { #2 } ]
5973         { #1 - \int_eval:n { \c@iRow - 1 } }
5974         { \exp_not:n { #3 } }
5975         [ \exp_not:n { #4 } ]
5976     }
5977 }
5978 }

```

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.

```

5979 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5980 {
5981     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5982     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5983     \seq_gclear:N \g_@@_rowlistcolors_seq
5984 }
5985 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5986 {
5987     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5988     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5989 }

```

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.

```

5990 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5991 {

```

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).

```

5992     \int_compare:nNtT { \c@jCol } > { \g_@@_col_total_int }
5993     {

```

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).

```

5994     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5995     {
5996         \exp_not:N \columncolor [ #1 ]
5997         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5998     }
5999 }
6000 }
6001 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6002 {
6003     \clist_map_inline:nn { #1 }
6004     {
6005         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6006         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6007         \columncolor { nocolor } { ##1 }
6008     }
6009 }

```

```

6010 \cs_new_protected:Npn \@@_EmptyRow:n #1
6011 {
6012   \clist_map_inline:nn { #1 }
6013   {
6014     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6015     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6016     \rowcolor { nocolor } { ##1 }
6017   }
6018 }

```

## 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`.

```

6019 \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`.

```

6020 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6021 {
6022   \int_if_zero:nTF { \l_@@_first_col_int }
6023   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6024   {
6025     \int_if_zero:nTF { \c_jCol }
6026     {
6027       \int_compare:nNnF { \c_iRow } = { -1 }
6028       {
6029         \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int - 1 }
6030         { #1 }
6031       }
6032     }
6033     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6034   }
6035 }

```

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.

```

6036 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6037 {
6038   \int_if_zero:nF { \c_iRow }
6039   {
6040     \int_compare:nNnF { \c_iRow } = { \l_@@_last_row_int }
6041     {
6042       \int_compare:nNnT { \c_jCol } > { \c_zero_int }
6043       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6044     }
6045   }
6046 }

```

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`.

```

6047 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6048 {
6049   \IfPackageLoadedTF { tikz }
6050   {
6051     \IfPackageLoadedTF { booktabs }
6052     { #2 }
6053     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6054   }
6055   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6056 }

6057 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6058 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6059 \cs_new:Npn \@@_TopRule_i:
6060 {
6061   \noalign \bgroup
6062   \peek_meaning:NTF [
6063   { \@@_TopRule_ii: }
6064   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6065 }

6066 \NewDocumentCommand \@@_TopRule_ii: { o }
6067 {
6068   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6069   {
6070     \@@_hline:n
6071     {
6072       position = \int_eval:n { \c@iRow + 1 } ,
6073       tikz =
6074       {
6075         line-width = #1 ,
6076         yshift = 0.25 \arrayrulewidth ,
6077         shorten-< = - 0.5 \arrayrulewidth
6078       } ,
6079       total-width = #1
6080     }
6081   }
6082   \skip_vertical:n { \belowrulesep + #1 }
6083   \egroup
6084 }

6085 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6086 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6087 \cs_new:Npn \@@_BottomRule_i:
6088 {
6089   \noalign \bgroup
6090   \peek_meaning:NTF [
6091   { \@@_BottomRule_ii: }
6092   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6093 }

6094 \NewDocumentCommand \@@_BottomRule_ii: { o }
6095 {
6096   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6097   {
6098     \@@_hline:n
6099     {
6100       position = \int_eval:n { \c@iRow + 1 } ,
6101       tikz =
6102       {
6103         line-width = #1 ,

```

```

6104         yshift = 0.25 \arrayrulewidth ,
6105         shorten~< = - 0.5 \arrayrulewidth
6106     } ,
6107     total-width = #1 ,
6108 }
6109 }
6110 \skip_vertical:N \aboverulesep
6111 \@@_create_row_node_i:
6112 \skip_vertical:n { #1 }
6113 \egroup
6114 }
6115 \NewExpandableDocumentCommand { \@@_MidRule } { }
6116 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6117 \cs_new:Npn \@@_MidRule_i:
6118 {
6119     \noalign \bgroup
6120     \peek_meaning:NTF [
6121         { \@@_MidRule_ii: }
6122         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6123     }
6124 \NewDocumentCommand \@@_MidRule_ii: { o }
6125 {
6126     \skip_vertical:N \aboverulesep
6127     \@@_create_row_node_i:
6128     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6129     {
6130         \@@_hline:n
6131         {
6132             position = \int_eval:n { \c@iRow + 1 } ,
6133             tikz =
6134             {
6135                 line-width = #1 ,
6136                 yshift = 0.25 \arrayrulewidth ,
6137                 shorten~< = - 0.5 \arrayrulewidth
6138             } ,
6139             total-width = #1 ,
6140         }
6141     }
6142     \skip_vertical:n { \belowrulesep + #1 }
6143     \egroup
6144 }

```

## 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.

```

6145 \keys_define:nn { nicematrix / Rules }
6146 {
6147     position .int_set:N = \l_@@_position_int ,
6148     position .value_required:n = true ,
6149     start .int_set:N = \l_@@_start_int ,
6150     end .code:n =
6151     \bool_lazy_or:nnTF
6152     { \tl_if_empty_p:n { #1 } }
6153     { \str_if_eq_p:ee { #1 } { last } }
6154     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6155     { \int_set:Nn \l_@@_end_int { #1 } }
6156 }

```

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 analysis 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.

```

6157 \keys_define:nn { nicematrix / RulesBis }
6158 {
6159     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6160     multiplicity .initial:n = 1 ,
6161     dotted .bool_set:N = \l_@@_dotted_bool ,
6162     dotted .initial:n = false ,
6163     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]`.

```

6164     color .code:n =
6165         \@@_set_CTarc:n { #1 }
6166         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6167     color .value_required:n = true ,
6168     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6169     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.

```

6170     tikz .code:n =
6171         \IfPackageLoadedTF { tikz }
6172         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6173         { \@@_error:n { tikz~without~tikz } } ,
6174     tikz .value_required:n = true ,
6175     total-width .dim_set:N = \l_@@_rule_width_dim ,
6176     total-width .value_required:n = true ,
6177     width .meta:n = { total-width = #1 } ,
6178     unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6179 }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6180 \cs_new_protected:Npn \@@_vline:n #1
6181 {

```

The group is for the options.

```

6182     \group_begin:
6183     \int_set_eq:NN \l_@@_end_int \c@iRow
6184     \keys_set_known:nnN { 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).

```

6185     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6186     \@@_vline_i:
6187     \group_end:
6188 }
6189 \cs_new_protected:Npn \@@_vline_i:
6190 {

```

`\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`.

```

6191     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6192     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int

```

```

6193 \l_tmpa_tl
6194 {

```

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.

```

6195 \bool_gset_true:N \g_tmpa_bool
6196 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6197 { \@@_test_vline_in_block:nnnnn ##1 }
6198 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6199 { \@@_test_vline_in_block:nnnnn ##1 }
6200 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6201 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6202 \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6203 \bool_if:NTF \g_tmpa_bool
6204 {
6205 \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.

```

6206 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6207 }
6208 {
6209 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6210 {
6211 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6212 \@@_vline_ii:
6213 \int_zero:N \l_@@_local_start_int
6214 }
6215 }
6216 }
6217 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6218 {
6219 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6220 \@@_vline_ii:
6221 }
6222 }

```

```

6223 \cs_new_protected:Npn \@@_test_in_corner_v:
6224 {
6225 \int_compare:nNnTF { \l_tmpb_tl } = { \c_jCol + 1 }
6226 {
6227 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6228 { \bool_set_false:N \g_tmpa_bool }
6229 }
6230 {
6231 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6232 {
6233 \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6234 { \bool_set_false:N \g_tmpa_bool }
6235 {
6236 \@@_if_in_corner:nT
6237 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6238 { \bool_set_false:N \g_tmpa_bool }
6239 }
6240 }
6241 }
6242 }

```

```

6243 \cs_new_protected:Npn \@@_vline_ii:
6244 {

```

```

6245 \tl_clear:N \l_@@_tikz_rule_tl
6246 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6247 \bool_if:NTF \l_@@_dotted_bool
6248 { \@@_vline_iv: }
6249 {
6250   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6251   { \@@_vline_iii: }
6252   { \@@_vline_v: }
6253 }
6254 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6255 \cs_new_protected:Npn \@@_vline_iii:
6256 {
6257   \pgfpicture
6258   \pgfrememberpicturepositiononpagetrue
6259   \pgf@relevantforpicturesizefalse
6260   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6261   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6262   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6263   \dim_set:Nn \l_tmpb_dim
6264   {
6265     \pgf@x
6266     - 0.5 \l_@@_rule_width_dim
6267     +
6268     ( \arrayrulewidth * \l_@@_multiplicity_int
6269       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6270   }
6271   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6272   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6273   \bool_lazy_all:nT
6274   {
6275     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6276     { \cs_if_exist_p:N \CT@drsc@ }
6277     { ! \tl_if_blank_p:o \CT@drsc@ }
6278   }
6279   {
6280     \group_begin:
6281     \CT@drsc@
6282     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6283     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6284     \dim_set:Nn \l_@@_tmpd_dim
6285     {
6286       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6287       * ( \l_@@_multiplicity_int - 1 )
6288     }
6289     \pgfpathrectanglecorners
6290     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6291     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6292     \pgfusepath { fill }
6293     \group_end:
6294   }
6295   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6296   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6297   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6298   {
6299     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6300     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6301     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6302   }
6303   \CT@arc@
6304   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6305   \pgfsetrectcap

```

```

6306     \pgfusepathqstroke
6307     \endpgfpicture
6308 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6309 \cs_new_protected:Npn \@@_vline_iv:
6310 {
6311     \pgfpicture
6312     \pgfrememberpicturepositiononpagetrue
6313     \pgf@relevantforpicturesizefalse
6314     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6315     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6316     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6317     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6318     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6319     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6320     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6321     \CT@arc@
6322     \@@_draw_line:
6323     \endpgfpicture
6324 }

```

The following code is for the case when the user uses the key `tikz`.

```

6325 \cs_new_protected:Npn \@@_vline_v:
6326 {
6327     \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).

```

6328     \CT@arc@
6329     \tl_if_empty:NF \l_@@_rule_color_tl
6330     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6331     \pgfrememberpicturepositiononpagetrue
6332     \pgf@relevantforpicturesizefalse
6333     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6334     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6335     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6336     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6337     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6338     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6339     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6340     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6341     ( \l_tmpb_dim , \l_tmpa_dim ) --
6342     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6343     \end { tikzpicture }
6344 }

```

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).

```

6345 \cs_new_protected:Npn \@@_draw_vlines:
6346 {
6347     \int_step_inline:nnn
6348     {
6349         \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6350         { 2 }
6351         { 1 }
6352     }
6353     {
6354         \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6355         { \c@jCol }
6356         { \int_eval:n { \c@jCol + 1 } }

```



```

6357     }
6358     {
6359         \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6360         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6361         { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6362     }
6363 }

```

## 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}`.

```

6364 \cs_new_protected:Npn \@@_hline:n #1
6365 {

```

The group is for the options.

```

6366     \group_begin:
6367     \int_set_eq:NN \l_@@_end_int \c@jCol
6368     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6369     \@@_hline_i:
6370     \group_end:
6371 }

6372 \cs_new_protected:Npn \@@_hline_i:
6373 {

```

`\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`.

```

6374     \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6375     \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6376     \l_tmpb_tl
6377     {

```

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.

```

6378         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6379         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6380         { \@@_test_hline_in_block:nnnn ##1 }

6381         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6382         { \@@_test_hline_in_block:nnnn ##1 }

6383         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6384         { \@@_test_hline_in_stroken_block:nnnn ##1 }

6385         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6386         \bool_if:NTF \g_tmpa_bool
6387         {
6388             \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.

```

6389             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6390         }
6391         {
6392             \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6393             {
6394                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6395                 \@@_hline_ii:
6396                 \int_zero:N \l_@@_local_start_int
6397             }
6398         }

```

```

6399     }
6400     \int_compare:nNtT { \l_@@_local_start_int } > { \c_zero_int }
6401     {
6402         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6403         \@@_hline_ii:
6404     }
6405 }

6406 \cs_new_protected:Npn \@@_test_in_corner_h:
6407 {
6408     \int_compare:nNtTF { \l_tmpa_tl } = { \c_iRow + 1 }
6409     {
6410         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6411         { \bool_set_false:N \g_tmpa_bool }
6412     }
6413     {
6414         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6415         {
6416             \int_compare:nNtTF { \l_tmpa_tl } = { \c_one_int }
6417             { \bool_set_false:N \g_tmpa_bool }
6418             {
6419                 \@@_if_in_corner:nT
6420                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6421                 { \bool_set_false:N \g_tmpa_bool }
6422             }
6423         }
6424     }
6425 }

6426 \cs_new_protected:Npn \@@_hline_ii:
6427 {
6428     \tl_clear:N \l_@@_tikz_rule_tl
6429     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6430     \bool_if:NTF \l_@@_dotted_bool
6431     { \@@_hline_iv: }
6432     {
6433         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6434         { \@@_hline_iii: }
6435         { \@@_hline_v: }
6436     }
6437 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6438 \cs_new_protected:Npn \@@_hline_iii:
6439 {
6440     \pgfpicture
6441     \pgfrememberpicturepositiononpagetrue
6442     \pgf@relevantforpicturesizefalse
6443     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6444     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6445     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6446     \dim_set:Nn \l_tmpb_dim
6447     {
6448         \pgf@y
6449         - 0.5 \l_@@_rule_width_dim
6450         +
6451         ( \arrayrulewidth * \l_@@_multiplicity_int
6452           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6453     }
6454     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6455     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```

```

6456 \bool_lazy_all:nT
6457 {
6458   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6459   { \cs_if_exist_p:N \CT@drsc@ }
6460   { ! \tl_if_blank_p:o \CT@drsc@ }
6461 }
6462 {
6463   \group_begin:
6464   \CT@drsc@
6465   \dim_set:Nn \l_@@_tmpd_dim
6466   {
6467     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6468     * ( \l_@@_multiplicity_int - 1 )
6469   }
6470   \pgfpathrectanglecorners
6471   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6472   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6473   \pgfusepathqfill
6474   \group_end:
6475 }
6476 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6477 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6478 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6479 {
6480   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6481   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6482   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6483 }
6484 \CT@arc@
6485 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6486 \pgfsetrectcap
6487 \pgfusepathqstroke
6488 \endpgfpicture
6489 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6490 \cs_new_protected:Npn \@@_hline_iv:
6491 {
6492   \pgfpicture
6493   \pgfrememberpicturepositiononpagetrue
6494   \pgf@relevantforpicturesizefalse
6495   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6496   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6497   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6498   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }

```

```

6499 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6500 \int_compare:nNt { \l_@@_local_start_int } = { \c_one_int }
6501 {
6502   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6503   \bool_if:NF \g_@@_delims_bool
6504   { \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.

```

6505   \tl_if_eq:NnF \g_@@_left_delim_tl (
6506     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6507   )
6508   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6509   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6510   \int_compare:nNt { \l_@@_local_end_int } = { \c_jCol }
6511   {
6512     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6513     \bool_if:NF \g_@@_delims_bool
6514     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6515     \tl_if_eq:NnF \g_@@_right_delim_tl )
6516     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6517   }
6518   \CT@arc@
6519   \@@_draw_line:
6520   \endpgfpicture
6521 }

```

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`).

```

6522 \cs_new_protected:Npn \@@_hline_v:
6523 {
6524   \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).

```

6525   \CT@arc@
6526   \tl_if_empty:NF \l_@@_rule_color_tl
6527   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6528   \pgfrememberpicturepositiononpagetrue
6529   \pgf@relevantforpicturesizefalse
6530   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6531   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6532   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6533   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6534   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6535   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6536   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6537   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6538     ( \l_tmpa_dim , \l_tmpb_dim ) --
6539     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6540   \end { tikzpicture }
6541 }

```

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).

```

6542 \cs_new_protected:Npn \@@_draw_hlines:
6543 {
6544   \int_step_inline:nnn
6545     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6546   {

```

```

6547     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6548     { \c@iRow }
6549     { \int_eval:n { \c@iRow + 1 } } }
6550   }
6551   {
6552     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6553     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6554     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6555   }
6556 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6557 \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.

```

6558 \cs_set:Npn \@@_Hline_i:n #1
6559 {
6560   \peek_remove_spaces:n
6561   {
6562     \peek_meaning:NTF \Hline
6563     { \@@_Hline_ii:nn { #1 + 1 } }
6564     { \@@_Hline_iii:n { #1 } }
6565   }
6566 }
6567 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6568 \cs_set:Npn \@@_Hline_iii:n #1
6569 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6570 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6571 {
6572   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6573   \skip_vertical:N \l_@@_rule_width_dim
6574   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6575   {
6576     \@@_hline:n
6577     {
6578       multiplicity = #1 ,
6579       position = \int_eval:n { \c@iRow + 1 } ,
6580       total-width = \dim_use:N \l_@@_rule_width_dim ,
6581       #2
6582     }
6583   }
6584   \egroup
6585 }

```

## 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`).

```

6586 \cs_new_protected:Npn \@@_custom_line:n #1
6587 {
6588   \str_clear_new:N \l_@@_command_str
6589   \str_clear_new:N \l_@@_ccommand_str
6590   \str_clear_new:N \l_@@_letter_str
6591   \tl_clear_new:N \l_@@_other_keys_tl
6592   \keys_set:known { 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.

```

6593 \bool_lazy_all:nTF
6594 {
6595   { \str_if_empty_p:N \l_@@_letter_str }
6596   { \str_if_empty_p:N \l_@@_command_str }
6597   { \str_if_empty_p:N \l_@@_ccommand_str }
6598 }
6599 { \@@_error:n { No~letter~and~no~command } }
6600 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6601 }
6602 \keys_define:nn { nicematrix / custom-line }
6603 {
6604   letter .str_set:N = \l_@@_letter_str ,
6605   letter .value_required:n = true ,
6606   command .str_set:N = \l_@@_command_str ,
6607   command .value_required:n = true ,
6608   ccommand .str_set:N = \l_@@_ccommand_str ,
6609   ccommand .value_required:n = true ,
6610 }

```

```

6611 \cs_new_protected:Npn \@@_custom_line_i:n #1
6612 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6613 \bool_set_false:N \l_@@_tikz_rule_bool
6614 \bool_set_false:N \l_@@_dotted_rule_bool
6615 \bool_set_false:N \l_@@_color_bool
6616 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6617 \bool_if:NT \l_@@_tikz_rule_bool
6618 {
6619   \IfPackageLoadedF { tikz }
6620   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6621   \bool_if:NT \l_@@_color_bool
6622   { \@@_error:n { color~in~custom~line~with~tikz } }
6623 }
6624 \bool_if:NT \l_@@_dotted_rule_bool
6625 {
6626   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6627   { \@@_error:n { key~multiplicity~with~dotted } }
6628 }
6629 \str_if_empty:NF \l_@@_letter_str
6630 {
6631   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6632   { \@@_error:n { Several~letters } }
6633   {
6634     \tl_if_in:NoTF
6635     \c_@@_forbidden_letters_str
6636     \l_@@_letter_str
6637     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6638   }

```

During the analysis of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6639 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6640 { \@@_v_custom_line:n { #1 } }
6641 }
6642 }
6643 }

```

```

6644 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6645 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6646 }
6647 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6648 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6649 \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.

```

6650 \keys_define:nn { nicematrix / custom-line-bis }
6651 {
6652   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6653   multiplicity .initial:n = 1 ,
6654   multiplicity .value_required:n = true ,
6655   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6656   color .value_required:n = true ,
6657   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6658   tikz .value_required:n = true ,
6659   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6660   dotted .value_forbidden:n = true ,
6661   total-width .code:n = { } ,
6662   total-width .value_required:n = true ,
6663   width .code:n = { } ,
6664   width .value_required:n = true ,
6665   sep-color .code:n = { } ,
6666   sep-color .value_required:n = true ,
6667   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6668 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6669 \bool_new:N \l_@@_dotted_rule_bool
6670 \bool_new:N \l_@@_tikz_rule_bool
6671 \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`.

```

6672 \keys_define:nn { nicematrix / custom-line-width }
6673 {
6674   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6675   multiplicity .initial:n = 1 ,
6676   multiplicity .value_required:n = true ,
6677   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6678   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6679   \bool_set_true:N \l_@@_total_width_bool ,
6680   total-width .value_required:n = true ,
6681   width .meta:n = { total-width = #1 } ,
6682   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6683 }

```

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`).

```

6684 \cs_new_protected:Npn \@@_h_custom_line:n #1
6685 {

```

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`).

```

6686   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6687   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6688 }

```

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`).

```
6689 \cs_new_protected:Npn \@@_c_custom_line:n #1
6690 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6691   \exp_args:Nc \NewExpandableDocumentCommand
6692     { nicematrix - \l_@@_ccommand_str }
6693     { 0 { } m }
6694     {
6695       \noalign
6696       {
6697         \@@_compute_rule_width:n { #1 , ##1 }
6698         \skip_vertical:n { \l_@@_rule_width_dim }
6699         \clist_map_inline:nn
6700           { ##2 }
6701           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6702       }
6703     }
6704   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6705 }
```

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*.

```
6706 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6707 {
6708   \tl_if_in:nnTF { #2 } { - }
6709     { \@@_cut_on_hyphen:w #2 \q_stop }
6710     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6711   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6712     {
6713       \@@_hline:n
6714       {
6715         #1 ,
6716         start = \l_tmpa_tl ,
6717         end = \l_tmpb_tl ,
6718         position = \int_eval:n { \c@iRow + 1 } ,
6719         total-width = \dim_use:N \l_@@_rule_width_dim
6720       }
6721     }
6722 }

6723 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6724 {
6725   \bool_set_false:N \l_@@_tikz_rule_bool
6726   \bool_set_false:N \l_@@_total_width_bool
6727   \bool_set_false:N \l_@@_dotted_rule_bool
6728   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6729   \bool_if:NF \l_@@_total_width_bool
6730   {
6731     \bool_if:NTF \l_@@_dotted_rule_bool
6732       { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6733       {
6734         \bool_if:NF \l_@@_tikz_rule_bool
6735         {
6736           \dim_set:Nn \l_@@_rule_width_dim
6737             {
6738               \arrayrulewidth * \l_@@_multiplicity_int
6739               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6740             }
6741         }
6742       }
6743   }
6744 }
```



```

6745 \cs_new_protected:Npn \@@_v_custom_line:n #1
6746 {
6747   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6748   \tl_gput_right:Ne \g_@@_array_preamble_tl
6749     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6750   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6751     {
6752     \@@_vline:n
6753       {
6754         #1 ,
6755         position = \int_eval:n { \c@jCol + 1 } ,
6756         total-width = \dim_use:N \l_@@_rule_width_dim
6757       }
6758     }
6759   \@@_rec_preamble:n
6760 }

6761 \@@_custom_line:n
6762 { 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.

```

6763 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6764 {
6765   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6766   {
6767     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6768     {
6769       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6770       {
6771         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6772         { \bool_gset_false:N \g_tmpa_bool }
6773       }
6774     }
6775   }
6776 }

```

The same for vertical rules.

```

6777 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6778 {
6779   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6780   {
6781     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6782     {
6783       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6784       {
6785         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6786         { \bool_gset_false:N \g_tmpa_bool }
6787       }
6788     }
6789   }
6790 }

6791 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6792 {
6793   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6794   {
6795     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6796     {

```

```

6797         \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6798         { \bool_gset_false:N \g_tmpa_bool }
6799         {
6800             \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6801             { \bool_gset_false:N \g_tmpa_bool }
6802         }
6803     }
6804 }
6805 }
6806 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6807 {
6808     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6809     {
6810         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6811         {
6812             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6813             { \bool_gset_false:N \g_tmpa_bool }
6814             {
6815                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6816                 { \bool_gset_false:N \g_tmpa_bool }
6817             }
6818         }
6819     }
6820 }

```

## 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.

```

6821 \cs_new_protected:Npn \@@_compute_corners:
6822 {
6823     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6824     { \@@_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`).

```

6825     \clist_clear:N \l_@@_corners_cells_clist
6826     \clist_map_inline:Nn \l_@@_corners_clist
6827     {
6828         \str_case:nnF { ##1 }
6829         {
6830             { NW }
6831             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6832             { NE }
6833             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6834             { SW }
6835             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6836             { SE }
6837             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6838         }
6839         { \@@_error:nn { bad~corner } { ##1 } }
6840     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6841     \clist_if_empty:NF \l_@@_corners_cells_clist
6842     {

```

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.

```

6843     \tl_gput_right:N\g_@@_aux_tl
6844     {
6845         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6846         { \l_@@_corners_cells_clist }
6847     }
6848 }
6849 }

6850 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6851 {
6852     \int_step_inline:nnn { #1 } { #3 }
6853     {
6854         \int_step_inline:nnn { #2 } { #4 }
6855         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6856     }
6857 }

6858 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6859 {
6860     \cs_if_exist:cTF
6861     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6862     { \prg_return_true: }
6863     { \prg_return_false: }
6864 }

```

“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.

```

6865 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6866 {

```

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.

```

6867     \bool_set_false:N \l_tmpa_bool
6868     \int_zero_new:N \l_@@_last_empty_row_int
6869     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6870     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6871     {
6872         \bool_lazy_or:nnTF
6873         {
6874             \cs_if_exist:pc
6875             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6876         }
6877         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6878         { \bool_set_true:N \l_tmpa_bool }
6879     }

```

```

6880         \bool_if:NF \l_tmpa_bool
6881         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6882     }
6883 }

```

Now, you determine the last empty cell in the row of number 1.

```

6884 \bool_set_false:N \l_tmpa_bool
6885 \int_zero_new:N \l_@@_last_empty_column_int
6886 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6887 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6888 {
6889     \bool_lazy_or:nnTF
6890     {
6891         \cs_if_exist_p:c
6892         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6893     }
6894     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6895     { \bool_set_true:N \l_tmpa_bool }
6896     {
6897         \bool_if:NF \l_tmpa_bool
6898         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6899     }
6900 }

```

Now, we loop over the rows.

```

6901 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6902 {

```

We treat the row number ##1 with another loop.

```

6903     \bool_set_false:N \l_tmpa_bool
6904     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6905     {
6906         \bool_lazy_or:nnTF
6907         { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6908         { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6909         { \bool_set_true:N \l_tmpa_bool }
6910         {
6911             \bool_if:NF \l_tmpa_bool
6912             {
6913                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6914                 \clist_put_right:Nn
6915                 \l_@@_corners_cells_clist
6916                 { ##1 - #####1 }
6917                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6918             }
6919         }
6920     }
6921 }
6922 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6923 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6924 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:  
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

## 24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
6925 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6926 \keys_define:nn { nicematrix / NiceMatrixBlock }
6927 {
6928   auto-columns-width .code:n =
6929   {
6930     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6931     \dim_gzero_new:N \g_@@_max_cell_width_dim
6932     \bool_set_true:N \l_@@_auto_columns_width_bool
6933   }
6934 }

6935 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6936 {
6937   \int_gincr:N \g_@@_NiceMatrixBlock_int
6938   \dim_zero:N \l_@@_columns_width_dim
6939   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6940   \bool_if:NT \l_@@_block_auto_columns_width_bool
6941   {
6942     \cs_if_exist:cT
6943     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6944     {
6945       \dim_set:Nn \l_@@_columns_width_dim
6946       {
6947         \use:c
6948         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6949       }
6950     }
6951   }
6952 }
```

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`).

```
6953 {
6954   \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.

```
6955   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6956   {
6957     \bool_if:NT \l_@@_block_auto_columns_width_bool
6958     {
6959       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6960       \iow_shipout:Ne \@mainaux
6961       {
6962         \cs_gset:cpn
6963         { @@ _ 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.

```
6964         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6965       }
6966       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6967     }
6968   }
6969   \ignorespacesafterend
6970 }
```

## 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).

```

6971 \cs_new_protected:Npn \@@_create_extra_nodes:
6972 {
6973   \bool_if:nTF \l_@@_medium_nodes_bool
6974   {
6975     \bool_if:NTF \l_@@_no_cell_nodes_bool
6976     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6977     {
6978       \bool_if:NTF \l_@@_large_nodes_bool
6979       \@@_create_medium_and_large_nodes:
6980       \@@_create_medium_nodes:
6981     }
6982   }
6983   {
6984     \bool_if:NT \l_@@_large_nodes_bool
6985     {
6986       \bool_if:NTF \l_@@_no_cell_nodes_bool
6987       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6988       \@@_create_large_nodes:
6989     }
6990   }
6991 }

```

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`.

```

6992 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6993 {
6994   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6995   {
6996     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
6997     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
6998     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
6999     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7000   }
7001   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7002   {
7003     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7004     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7005     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7006     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7007   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7008 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7009 {
7010     \int_step_variable:nnNn
7011     \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.

```

7012 {
7013     \cs_if_exist:cT
7014     { 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`.

```

7015 {
7016     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7017     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7018     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7019     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7020     {
7021         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7022         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7023     }

```

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`.

```

7024     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7025     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7026     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7027     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7028     {
7029         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7030         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7031     }
7032 }
7033 }
7034 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7035 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7036 {
7037     \dim_compare:nNnT
7038     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7039     {
7040         \@@_qpoint:n { row - \@@_i: - base }
7041         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7042         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7043     }
7044 }
7045 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7046 {
7047     \dim_compare:nNnT
7048     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7049     {
7050         \@@_qpoint:n { col - \@@_j: }
7051         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7052         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7053     }
7054 }
7055 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7056 \cs_new_protected:Npn \@@_create_medium_nodes:
7057 {
7058   \pgfpicture
7059     \pgfrememberpicturerepositiononpagetrue
7060     \pgf@relevantforpicturesizefalse
7061     \@@_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”.

```

7062     \tl_set:Nn \l_@@_suffix_tl { -medium }
7063     \@@_create_nodes:
7064   \endpgfpicture
7065 }

```

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:`.

```

7066 \cs_new_protected:Npn \@@_create_large_nodes:
7067 {
7068   \pgfpicture
7069     \pgfrememberpicturerepositiononpagetrue
7070     \pgf@relevantforpicturesizefalse
7071     \@@_computations_for_medium_nodes:
7072     \@@_computations_for_large_nodes:
7073     \tl_set:Nn \l_@@_suffix_tl { - large }
7074     \@@_create_nodes:
7075   \endpgfpicture
7076 }

7077 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7078 {
7079   \pgfpicture
7080     \pgfrememberpicturerepositiononpagetrue
7081     \pgf@relevantforpicturesizefalse
7082     \@@_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”.

```

7083     \tl_set:Nn \l_@@_suffix_tl { - medium }
7084     \@@_create_nodes:
7085     \@@_computations_for_large_nodes:
7086     \tl_set:Nn \l_@@_suffix_tl { - large }
7087     \@@_create_nodes:
7088   \endpgfpicture
7089 }

```

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.

```

7090 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7091 {
7092   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7093   \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`.

```

7094   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7095   {
7096     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

---

<sup>15</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`



```

7097     {
7098     (
7099         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7100         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7101     )
7102     / 2
7103     }
7104     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7105     { l_@@_row _ \@@_i: _ min_dim }
7106 }
7107 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7108 {
7109     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7110     {
7111     (
7112         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7113         \dim_use:c
7114         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7115     )
7116     / 2
7117     }
7118     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7119     { l_@@_column _ \@@_j: _ max _ dim }
7120 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7121     \dim_sub:cn
7122     { l_@@_column _ 1 _ min _ dim }
7123     \l_@@_left_margin_dim
7124     \dim_add:cn
7125     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7126     \l_@@_right_margin_dim
7127 }

```

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).

```

7128 \cs_new_protected:Npn \@@_create_nodes:
7129 {
7130     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7131     {
7132         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7133         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7134         \@@_pgf_rect_node:nnnnn
7135         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7136         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7137         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7138         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7139         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7140         \str_if_empty:NF \l_@@_name_str
7141         {
7142             \pgfnodealias
7143             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7144             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7145         }
7146     }
7147 }
7148 \int_step_inline:nn { \c@iRow }

```

```

7149 {
7150   \pgfnodealias
7151   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7152   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7153 }
7154 \int_step_inline:nn { \c@jCol }
7155 {
7156   \pgfnodealias
7157   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7158   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7159 }
7160 \pgfnodealias % added 2025-04-05
7161 { \@@_env: - last - last \l_@@_suffix_tl }
7162 { \@@_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  $n$ .

```

7163   \seq_map_pairwise_function:NNN
7164   \g_@@_multicolumn_cells_seq
7165   \g_@@_multicolumn_sizes_seq
7166   \@@_node_for_multicolumn:nn
7167 }

7168 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7169 {
7170   \cs_set_nopar:Npn \@@_i: { #1 }
7171   \cs_set_nopar:Npn \@@_j: { #2 }
7172 }

```

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”).

```

7173 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7174 {
7175   \@@_extract_coords_values: #1 \q_stop
7176   \@@_pgf_rect_node:nnnnn
7177   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7178   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7179   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7180   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7181   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7182   \str_if_empty:NF \l_@@_name_str
7183   {
7184     \pgfnodealias
7185     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7186     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7187   }
7188 }

```

## 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).

```

7189 \keys_define:nn { nicematrix / Block / FirstPass }
7190 {
7191   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7192     \bool_set_true:N \l_@@_p_block_bool ,
7193   j .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   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7201   L .value_forbidden:n = true ,
7202   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7203   R .value_forbidden:n = true ,
7204   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7205   C .value_forbidden:n = true ,
7206   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7207   t .value_forbidden:n = true ,
7208   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7209   T .value_forbidden:n = true ,
7210   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7211   b .value_forbidden:n = true ,
7212   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7213   B .value_forbidden:n = true ,
7214   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7215   m .value_forbidden:n = true ,
7216   v-center .meta:n = m ,
7217   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7218   p .value_forbidden:n = true ,
7219   color .code:n =
7220     \@@_color:n { #1 }
7221     \tl_set_rescan:Nnn
7222       \l_@@_draw_tl
7223       { \char_set_catcode_other:N ! }
7224       { #1 } ,
7225   color .value_required:n = true ,
7226   respect-arraystretch .code:n =
7227     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7228   respect-arraystretch .value_forbidden:n = true ,
7229 }

```

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.

```

7230 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7231 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7232 {

```

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).

```

7233   \tl_if_blank:nTF { #2 }
7234   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7235   {
7236     \tl_if_in:nnTF { #2 } { - }
7237     {
7238       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7239       \@@_Block_i_czech:w \@@_Block_i:w
7240       #2 \q_stop
7241     }
7242     {
7243       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7244         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7245     }
7246 }
7247 { #1 } { #3 } { #4 }
7248 \ignorespaces
7249 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7250 \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`.

```

7251 {
7252   \char_set_catcode_active:N -
7253   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7254 }

```

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.

```

7255 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7256 {

```

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).

```

7257   \bool_lazy_or:nnTF
7258     { \tl_if_blank_p:n { #1 } }
7259     { \str_if_eq_p:ee { * } { #1 } }
7260     { \int_set:Nn \l_tmpa_int { 100 } }
7261     { \int_set:Nn \l_tmpa_int { #1 } }
7262   \bool_lazy_or:nnTF
7263     { \tl_if_blank_p:n { #2 } }
7264     { \str_if_eq_p:ee { * } { #2 } }
7265     { \int_set:Nn \l_tmpb_int { 100 } }
7266     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7267   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7268   {
7269     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7270     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7271     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7272   }
7273   { \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.

```

7274   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7275   \tl_set:Ne \l_tmpa_tl
7276   {
7277     { \int_use:N \c@iRow }
7278     { \int_use:N \c@jCol }
7279     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7280     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7281   }

```

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).

```

7282 \bool_set_false:N \l_tmpa_bool
7283 \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7284 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7285 \bool_case:nF
7286 {
7287     \l_tmpa_bool { \@@_Block_vii:eennn }
7288     \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.

```

7289 \l_@@_X_bool { \@@_Block_v:eennn }
7290 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7291 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7292 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7293 }
7294 { \@@_Block_v:eennn }
7295 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7296 }

```

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.

```

7297 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7298 {
7299     \int_gincr:N \g_@@_block_box_int
7300     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7301     {
7302         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7303         {
7304             \@@_actually_diagbox:nnnnnn
7305             { \int_use:N \c@iRow }
7306             { \int_use:N \c@jCol }
7307             { \int_eval:n { \c@iRow + #1 - 1 } }
7308             { \int_eval:n { \c@jCol + #2 - 1 } }
7309             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7310             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7311         }
7312     }
7313     \box_gclear_new:c
7314     { 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!

```

7315 \hbox_gset:cn
7316 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7317 {

```

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`).

```

7318 \tl_if_empty:NTF \l_@@_color_tl
7319 { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7320 { \@@_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`.

```

7321 \int_compare:nNnT { #1 } = { \c_one_int }
7322 {
7323 \int_if_zero:nTF { \c_iRow }
7324 {

```

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}$

```

```

7325 \cs_set_eq:NN \Block \@@_NullBlock:
7326 \l_@@_code_for_first_row_tl
7327 }
7328 {
7329 \int_compare:nNnT { \c_iRow } = { \l_@@_last_row_int }
7330 {
7331 \cs_set_eq:NN \Block \@@_NullBlock:
7332 \l_@@_code_for_last_row_tl
7333 }
7334 }
7335 \g_@@_row_style_tl
7336 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7337 \@@_reset_arraystretch:
7338 \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7339 #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.).

```

7340 \@@_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}`.

```

7341     \bool_if:NTF \l_@@_tabular_bool
7342     {
7343         \bool_lazy_all:nTF
7344         {
7345             { \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.

```

7346         {
7347             ! \dim_compare_p:nNn
7348             { \l_@@_col_width_dim } < { \c_zero_dim }
7349         }
7350         { ! \g_@@_rotate_bool }
7351     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7352     {
7353         \use:e
7354         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7355             \exp_not:N \begin { minipage }
7356             [ \str_lowercase:f \l_@@_vpos_block_str ]
7357             { \l_@@_col_width_dim }
7358             \str_case:on \l_@@_hpos_block_str
7359             { c \centering r \raggedleft l \raggedright }
7360         }
7361         #5
7362     \end { minipage }
7363 }

```

In the other cases, we use a `{tabular}`.

```

7364     {
7365         \use:e
7366         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7367             \exp_not:N \begin { tabular }
7368             [ \str_lowercase:f \l_@@_vpos_block_str ]
7369             { @ { } \l_@@_hpos_block_str @ { } }
7370         }
7371         #5
7372     \end { tabular }
7373 }
7374 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7375     {
7376         \c_math_toggle_token
7377         \use:e
7378         {

```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7379             \exp_not:N \begin { array }
7380             [ \str_lowercase:f \l_@@_vpos_block_str ]
7381             { @ { } \l_@@_hpos_block_str @ { } }
7382         }
7383         #5
7384     \end { array }

```

```

7385         \c_math_toggle_token
7386     }
7387 }

```

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).

```

7388     \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.

```

7389     \int_compare:nNnT { #2 } = { \c_one_int }
7390     {
7391         \dim_gset:Nn \g_@@_blocks_wd_dim
7392         {
7393             \dim_max:nn
7394             { \g_@@_blocks_wd_dim }
7395             {
7396                 \box_wd:c
7397                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7398             }
7399         }
7400     }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7401     \int_compare:nNnT { #1 } = { \c_one_int }
7402     {
7403         \bool_lazy_any:nT
7404         {
7405             { \str_if_empty_p:N \l_@@_vpos_block_str }
7406             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7407             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7408         }
7409         { \@@_adjust_blocks_ht_dp: }
7410     }
7411     \seq_gput_right:Ne \g_@@_blocks_seq
7412     {
7413         \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.

```

7414     {
7415         \exp_not:n { #3 } ,
7416         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7417     \bool_if:NT \g_@@_rotate_bool
7418     {
7419         \bool_if:NTF \g_@@_rotate_c_bool
7420         { m }
7421         {
7422             \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7423             { T }
7424         }
7425     }
7426 }
7427 {
7428     \box_use_drop:c
7429     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```



```

7430     }
7431   }
7432   \bool_set_false:N \g_@@_rotate_c_bool
7433 }

7434 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7435 {
7436   \dim_gset:Nn \g_@@_blocks_ht_dim
7437   {
7438     \dim_max:nn
7439     { \g_@@_blocks_ht_dim }
7440     {
7441       \box_ht:c
7442       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7443     }
7444   }
7445   \dim_gset:Nn \g_@@_blocks_dp_dim
7446   {
7447     \dim_max:nn
7448     { \g_@@_blocks_dp_dim }
7449     {
7450       \box_dp:c
7451       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7452     }
7453   }
7454 }

7455 \cs_new:Npn \@@_adjust_hpos_rotate:
7456 {
7457   \bool_if:NT \g_@@_rotate_bool
7458   {
7459     \str_set:Ne \l_@@_hpos_block_str
7460     {
7461       \bool_if:NTF \g_@@_rotate_c_bool
7462       { c }
7463       {
7464         \str_case:onF \l_@@_vpos_block_str
7465         { b l B l t r T r }
7466         {
7467           \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7468           { r }
7469           { l }
7470         }
7471       }
7472     }
7473   }
7474 }
7475 \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*.

```

7476 \cs_new_protected:Npn \@@_rotate_box_of_block:
7477 {
7478   \box_grotate:cn
7479   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7480   { 90 }
7481   \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7482   {
7483     \vbox_gset_top:cn
7484     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7485     {
7486       \skip_vertical:n { 0.8 ex }

```

```

7487         \box_use:c
7488         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7489     }
7490 }
7491 \bool_if:NT \g_@@_rotate_c_bool
7492 {
7493     \hbox_gset:cn
7494     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7495     {
7496         \c_math_toggle_token
7497         \vcenter
7498         {
7499             \box_use:c
7500             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7501         }
7502         \c_math_toggle_token
7503     }
7504 }
7505 }

```

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.

```

7506 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7507 {
7508     \seq_gput_right:Ne \g_@@_blocks_seq
7509     {
7510         \l_tmpa_tl
7511         { \exp_not:n { #3 } }
7512         {
7513             \bool_if:NTF \l_@@_tabular_bool
7514             {
7515                 \group_begin:

```

The following command 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

```

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).

```

7521         \IfPackageLoadedTF { latex-lab-testphase-table }
7522         { \tag_stop:n { table } }
7523         \use:e
7524         {
7525             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7526             { @ { } \l_@@_hpos_block_str @ { } }
7527         }
7528         #5
7529         \end { tabular }
7530     }
7531     \group_end:
7532 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7533     {
7534         \group_begin:
The following will be no-op when respect-arraystretch is in force.
7535         \@@_reset_arraystretch:
7536         \exp_not:n
7537         {
7538             \dim_zero:N \extrarowheight
7539             #4
7540             \c_math_toggle_token
7541             \use:e
7542             {
7543                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7544                 { @ { } \l_@@_hpos_block_str @ { } }
7545             }
7546             #5
7547             \end { array }
7548             \c_math_toggle_token
7549         }
7550         \group_end:
7551     }
7552 }
7553 }
7554 }
7555 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7556 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7557 {
7558     \seq_gput_right:Ne \g_@@_blocks_seq
7559     {
7560         \l_tmpa_tl
7561         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7562         { { \exp_not:n { #4 #5 } } }
7563     }
7564 }
7565 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7566 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7567 {
7568     \seq_gput_right:Ne \g_@@_blocks_seq
7569     {
7570         \l_tmpa_tl
7571         { \exp_not:n { #3 } }
7572         { \exp_not:n { #4 #5 } }
7573     }
7574 }
7575 \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).

```

7576 \keys_define:nn { nicematrix / Block / SecondPass }
7577 {
7578     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7579     ampersand-in-blocks .default:n = true ,
7580     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7581   tikz .code:n =
7582       \IfPackageLoadedTF { tikz }
7583       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7584       { \@@_error:n { tikz~key~without~tikz } } ,
7585   tikz .value_required:n = true ,
7586   fill .code:n =
7587       \tl_set_rescan:Nnn
7588       \l_@@_fill_tl
7589       { \char_set_catcode_other:N ! }
7590       { #1 } ,
7591   fill .value_required:n = true ,
7592   opacity .tl_set:N = \l_@@_opacity_tl ,
7593   opacity .value_required:n = true ,
7594   draw .code:n =
7595       \tl_set_rescan:Nnn
7596       \l_@@_draw_tl
7597       { \char_set_catcode_other:N ! }
7598       { #1 } ,
7599   draw .default:n = default ,
7600   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7601   rounded-corners .default:n = 4 pt ,
7602   color .code:n =
7603       \@@_color:n { #1 }
7604       \tl_set_rescan:Nnn
7605       \l_@@_draw_tl
7606       { \char_set_catcode_other:N ! }
7607       { #1 } ,
7608   borders .clist_set:N = \l_@@_borders_clist ,
7609   borders .value_required:n = true ,
7610   hvlines .meta:n = { vlines , hlines } ,
7611   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7612   vlines .default:n = true ,
7613   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7614   hlines .default:n = true ,
7615   line-width .dim_set:N = \l_@@_line_width_dim ,
7616   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7617   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7618       \bool_set_true:N \l_@@_p_block_bool ,
7619   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7620   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7621   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7622   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7623       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7624   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7625       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7626   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7627       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7628   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7629   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7630   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7631   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7632   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7633   m .value_forbidden:n = true ,
7634   v-center .meta:n = m ,
7635   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7636   p .value_forbidden:n = true ,
7637   name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7638   name .value_required:n = true ,
7639   name .initial:n = ,
7640   respect-arraystretch .code:n =
7641       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,

```

```

7642     respect-arraystretch .value_forbidden:n = true ,
7643     transparent .bool_set:N = \l_@@_transparent_bool ,
7644     transparent .default:n = true ,
7645     transparent .initial:n = false ,
7646     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7647 }

```

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.

```

7648 \cs_new_protected:Npn \@@_draw_blocks:
7649 {
7650   \bool_if:NTF \c_@@_revtex_bool
7651     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7652     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7653   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7654 }
7655 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7656 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7657   \int_zero:N \l_@@_last_row_int
7658   \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”).

```

7659   \int_compare:nNnTF { #3 } > { 98 }
7660     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7661     { \int_set:Nn \l_@@_last_row_int { #3 } }
7662   \int_compare:nNnTF { #4 } > { 98 }
7663     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7664     { \int_set:Nn \l_@@_last_col_int { #4 } }
7665   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7666     {
7667       \bool_lazy_and:nnTF
7668         { \l_@@_preamble_bool }
7669         {
7670           \int_compare_p:n
7671             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7672         }
7673       {
7674         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7675         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7676         \@@_msg_redirect_name:nn { columns-not-used } { none }
7677       }
7678       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7679     }
7680   {
7681     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7682       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7683       {
7684         \@@_Block_v:nneenn
7685           { #1 }
7686           { #2 }
7687           { \int_use:N \l_@@_last_row_int }
7688           { \int_use:N \l_@@_last_col_int }
7689           { #5 }

```

```

7690         { #6 }
7691     }
7692 }
7693 }

```

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

```

7694 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7695 {

```

The group is for the keys.

```

7696     \group_begin:
7697     \int_compare:nNnT { #1 } = { #3 }
7698     { \str_set:Nn \l_@@_vpos_block_str { t } }
7699     \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 than `\str_if_in:nnT`.

```

7700     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7701     \bool_lazy_and:nnT
7702     { \l_@@_vlines_block_bool }
7703     { ! \l_@@_ampersand_bool }
7704     {
7705         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7706         {
7707             \@@_vlines_block:nnn
7708             { \exp_not:n { #5 } }
7709             { #1 - #2 }
7710             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7711         }
7712     }
7713     \bool_if:NT \l_@@_hlines_block_bool
7714     {
7715         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7716         {
7717             \@@_hlines_block:nnn
7718             { \exp_not:n { #5 } }
7719             { #1 - #2 }
7720             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7721         }
7722     }
7723     \bool_if:NF \l_@@_transparent_bool
7724     {
7725         \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7726         {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7727         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7728         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7729     }
7730 }

```

```

7731 \tl_if_empty:NF \l_@@_draw_tl
7732 {
7733     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7734     { \@@_error:n { hlines-with-color } }
7735     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7736     {
7737         \@@_stroke_block:nnn

```

#5 are the options

```

7738         { \exp_not:n { #5 } }
7739         { #1 - #2 }
7740         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7741     }
7742     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7743     { { #1 } { #2 } { #3 } { #4 } }
7744 }
7745 \clist_if_empty:NF \l_@@_borders_clist
7746 {
7747     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7748     {
7749         \@@_stroke_borders_block:nnn
7750         { \exp_not:n { #5 } }
7751         { #1 - #2 }
7752         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7753     }
7754 }
7755 \tl_if_empty:NF \l_@@_fill_tl
7756 {
7757     \@@_add_opacity_to_fill:
7758     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7759     {
7760         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7761         { #1 - #2 }
7762         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7763         { \dim_use:N \l_@@_rounded_corners_dim }
7764     }
7765 }
7766 \seq_if_empty:NF \l_@@_tikz_seq
7767 {
7768     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7769     {
7770         \@@_block_tikz:nnnnn
7771         { \seq_use:Nn \l_@@_tikz_seq { , } }
7772         { #1 }
7773         { #2 }
7774         { \int_use:N \l_@@_last_row_int }
7775         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7776     }
7777 }
7778 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7779 {
7780     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7781     {
7782         \@@_actually_diagbox:nnnnnn
7783         { #1 }
7784         { #2 }
7785         { \int_use:N \l_@@_last_row_int }
7786         { \int_use:N \l_@@_last_col_int }
7787         { \exp_not:n { ##1 } }
7788         { \exp_not:n { ##2 } }
7789     }
7790 }

```

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.

```

7791 \pgfpicture
7792 \pgfrememberpicturepositiononpagetrue
7793 \pgf@relevantforpicturesizefalse
7794 \@@_qpoint:n { row - #1 }
7795 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7796 \@@_qpoint:n { col - #2 }
7797 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7798 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7799 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7800 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7801 \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.

```

7802 \@@_pgf_rect_node:nnnnn
7803 { \@@_env: - #1 - #2 - block }
7804 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7805 \str_if_empty:NF \l_@@_block_name_str
7806 {
7807   \pgfnodealias
7808   { \@@_env: - \l_@@_block_name_str }
7809   { \@@_env: - #1 - #2 - block }
7810   \str_if_empty:NF \l_@@_name_str
7811   {
7812     \pgfnodealias
7813     { \l_@@_name_str - \l_@@_block_name_str }
7814     { \@@_env: - #1 - #2 - block }
7815   }
7816 }

```

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.

```

7817 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7818 {
7819   \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.

```

7820 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7821 {

```

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.

```

7822 \cs_if_exist:cT
7823 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```



```

7824     {
7825         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7826         {
7827             \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7828             \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7829         }
7830     }
7831 }

```

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.

```

7832     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7833     {
7834         \@@_qpoint:n { col - #2 }
7835         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7836     }
7837     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7838     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7839     {
7840         \cs_if_exist:cT
7841         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7842         {
7843             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7844             {
7845                 \pgfpointanchor
7846                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7847                 { east }
7848                 \dim_set:Nn \l_@@_tmpd_dim
7849                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7850             }
7851         }
7852     }
7853     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7854     {
7855         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7856         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7857     }
7858     \@@_pgf_rect_node:nnnnn
7859     { \@@_env: - #1 - #2 - block - short }
7860     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7861 }

```

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.

```

7862     \bool_if:NT \l_@@_medium_nodes_bool
7863     {
7864         \@@_pgf_rect_node:nnn
7865         { \@@_env: - #1 - #2 - block - medium }
7866         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7867         {
7868             \pgfpointanchor
7869             { \@@_env:
7870               - \int_use:N \l_@@_last_row_int
7871               - \int_use:N \l_@@_last_col_int - medium
7872             }
7873             { south-east }
7874         }
7875     }
7876     \endpgfpicture
7877
7878     \bool_if:NTF \l_@@_ampersand_bool
7879     {

```

```

7880 \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7881 \int_zero_new:N \l_@@_split_int
7882 \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7883 \pgfpicture
7884 \pgfrememberpicturepositiononpagetrue
7885 \pgf@relevantforpicturesizefalse
7886
7887 \@@_qpoint:n { row - #1 }
7888 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7889 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7890 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7891 \@@_qpoint:n { col - #2 }
7892 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7893 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7894 \dim_set:Nn \l_tmpb_dim
7895 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7896 \bool_lazy_or:nnT
7897 { \l_@@_vlines_block_bool }
7898 { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
7899 {
7900   \int_step_inline:nn { \l_@@_split_int - 1 }
7901   {
7902     \pgfpathmoveto
7903     {
7904       \pgfpoint
7905       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7906       \l_@@_tmpc_dim
7907     }
7908     \pgfpathlineto
7909     {
7910       \pgfpoint
7911       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7912       \l_@@_tmpd_dim
7913     }
7914     \CT@arc@
7915     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7916     \pgfsetrectcap
7917     \pgfusepathqstroke
7918   }
7919 }
7920 \@@_qpoint:n { row - #1 - base }
7921 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7922 \int_step_inline:nn { \l_@@_split_int }
7923 {
7924   \group_begin:
7925   \dim_set:Nn \col@sep
7926   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7927   \pgftransformshift
7928   {
7929     \pgfpoint
7930     {
7931       \l_tmpa_dim + ##1 \l_tmpb_dim -
7932       \str_case:on \l_@@_hpos_block_str
7933       {
7934         l { \l_tmpb_dim + \col@sep }
7935         c { 0.5 \l_tmpb_dim }
7936         r { \col@sep }
7937       }
7938     }
7939     { \l_@@_tmpc_dim }
7940   }
7941   \pgfset { inner~sep = \c_zero_dim }
7942   \pgfnode

```

```

7943     { rectangle }
7944     {
7945         \str_case:on \l_@@_hpos_block_str
7946         {
7947             c { base }
7948             l { base~west }
7949             r { base~east }
7950         }
7951     }
7952     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7953     \group_end:
7954 }
7955 \endpgfpicture
7956 }

```

Now the case where there is no ampersand & in the content of the block.

```

7957 {
7958     \bool_if:NTF \l_@@_p_block_bool
7959     {

```

When the final user has used the key p, we have to compute the width.

```

7960         \pgfpicture
7961         \pgfrememberpicturepositiononpagetrue
7962         \pgf@relevantforpicturesizefalse
7963         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7964         {
7965             \@@_qpoint:n { col - #2 }
7966             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7967             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7968         }
7969         {
7970             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7971             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7972             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7973         }
7974         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7975     \endpgfpicture
7976     \hbox_set:Nn \l_@@_cell_box
7977     {
7978         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7979         { \g_tmpb_dim }
7980         \str_case:on \l_@@_hpos_block_str
7981         { c \centering r \raggedleft l \raggedright j { } }
7982         #6
7983         \end { minipage }
7984     }
7985 }
7986 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7987 \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.

```

7988 \pgfpicture
7989 \pgfrememberpicturepositiononpagetrue
7990 \pgf@relevantforpicturesizefalse
7991 \bool_lazy_any:nTF
7992 {
7993     { \str_if_empty_p:N \l_@@_vpos_block_str }
7994     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
7995     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
7996     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
7997 }
7998 {

```

If we are in the first column, we must put the block as if it was with the key r.

```
7999      \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.

```
8000      \bool_if:nT \g_@@_last_col_found_bool
8001      {
8002          \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8003          { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8004      }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
8005      \tl_set:Ne \l_tmpa_tl
8006      {
8007          \str_case:on \l_@@_vpos_block_str
8008          {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
8009          { } {
8010              \str_case:on \l_@@_hpos_block_str
8011              {
8012                  c { center }
8013                  l { west }
8014                  r { east }
8015                  j { center }
8016              }
8017          }
8018      c {
8019          \str_case:on \l_@@_hpos_block_str
8020          {
8021              c { center }
8022              l { west }
8023              r { east }
8024              j { center }
8025          }
8026      }
8027  }
8028  T {
8029      \str_case:on \l_@@_hpos_block_str
8030      {
8031          c { north }
8032          l { north-west }
8033          r { north-east }
8034          j { north }
8035      }
8036  }
8037  }
8038  B {
8039      \str_case:on \l_@@_hpos_block_str
8040      {
8041          c { south }
8042          l { south-west }
8043          r { south-east }
8044          j { south }
8045      }
8046  }
8047  }
8048  }
8049  }
8050  \pgftransformshift
8051  {
8052      \pgfpointanchor
8053      {
8054          \@@_env: - #1 - #2 - block
```

```

8055         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8056     }
8057     { \l_tmpa_tl }
8058 }
8059 \pgfset { inner~sep = \c_zero_dim }
8060 \pgfnode
8061 { rectangle }
8062 { \l_tmpa_tl }
8063 { \box_use_drop:N \l_@@_cell_box } { } { }
8064 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

8065 {
8066     \pgfextracty \l_tmpa_dim
8067     {
8068         \@@_qpoint:n
8069         {
8070             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8071             - base
8072         }
8073     }
8074     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

8075     \pgfpointanchor
8076     {
8077         \@@_env: - #1 - #2 - block
8078         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8079     }
8080     {
8081         \str_case:on \l_@@_hpos_block_str
8082         {
8083             c { center }
8084             l { west }
8085             r { east }
8086             j { center }
8087         }
8088     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8089     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8090     \pgfset { inner~sep = \c_zero_dim }
8091     \pgfnode
8092     { rectangle }
8093     {
8094         \str_case:on \l_@@_hpos_block_str
8095         {
8096             c { base }
8097             l { base~west }
8098             r { base~east }
8099             j { base }
8100         }
8101     }
8102     { \box_use_drop:N \l_@@_cell_box } { } { }
8103 }
8104 \endpgfpicture
8105 }
8106 \group_end:
8107 }
8108 \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).

```

8109 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8110 {
8111   \pgfpicture
8112   \pgfrememberpicturepositiononpagetrue
8113   \pgf@relevantforpicturesizefalse
8114   \pgfpathrectanglecorners
8115     { \pgfpoint { #2 } { #3 } }
8116     { \pgfpoint { #4 } { #5 } }
8117   \pgfsetfillcolor { #1 }
8118   \pgfusepath { fill }
8119   \endpgfpicture
8120 }

```

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).

```

8121 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8122 {
8123   \tl_if_empty:NF \l_@@_opacity_tl
8124   {
8125     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8126       {
8127         \tl_set:Nc \l_@@_fill_tl
8128           {
8129             [ opacity = \l_@@_opacity_tl ,
8130             \tl_tail:o \l_@@_fill_tl
8131           }
8132       }
8133       {
8134         \tl_set:Nc \l_@@_fill_tl
8135           { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8136       }
8137     }
8138   }

```

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).

```

8139 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8140 {
8141   \group_begin:
8142   \tl_clear:N \l_@@_draw_tl
8143   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8144   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8145   \pgfpicture
8146   \pgfrememberpicturepositiononpagetrue
8147   \pgf@relevantforpicturesizefalse
8148   \tl_if_empty:NF \l_@@_draw_tl
8149   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8150     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8151       { \CT@arc@ }
8152       { \@@_color:o \l_@@_draw_tl }
8153   }
8154   \pgfsetcornersarced
8155   {
8156     \pgfpoint
8157       { \l_@@_rounded_corners_dim }
8158       { \l_@@_rounded_corners_dim }
8159   }

```

```

8160 \@@_cut_on_hyphen:w #2 \q_stop
8161 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8162 {
8163   \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8164   {
8165     \@@_qpoint:n { row - \l_tmpa_tl }
8166     \dim_set_eq:NN \l_tmpb_dim \pgf@y
8167     \@@_qpoint:n { col - \l_tmpb_tl }
8168     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8169     \@@_cut_on_hyphen:w #3 \q_stop
8170     \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8171     { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8172     \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8173     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8174     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8175     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8176     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8177     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8178     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8179     \pgfpathrectanglecorners
8180     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8181     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8182     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8183     { \pgfusepathqstroke }
8184     { \pgfusepath { stroke } }
8185   }
8186 }
8187 \endpgfpicture
8188 \group_end:
8189 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8190 \keys_define:nn { nicematrix / BlockStroke }
8191 {
8192   color .tl_set:N = \l_@@_draw_tl ,
8193   draw .code:n =
8194     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8195   draw .default:n = default ,
8196   line-width .dim_set:N = \l_@@_line_width_dim ,
8197   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8198   rounded-corners .default:n = 4 pt
8199 }

```

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).

```

8200 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8201 {
8202   \group_begin:
8203   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8204   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8205   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8206   \@@_cut_on_hyphen:w #2 \q_stop
8207   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8208   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8209   \@@_cut_on_hyphen:w #3 \q_stop
8210   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8211   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8212   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8213   {
8214     \use:e
8215     {
8216       \@@_vline:n

```

```

8217         {
8218             position = ##1 ,
8219             start = \l_@@_tmpc_tl ,
8220             end = \int_eval:n { \l_tmpa_tl - 1 } ,
8221             total-width = \dim_use:N \l_@@_line_width_dim
8222         }
8223     }
8224 }
8225 \group_end:
8226 }
8227 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8228 {
8229     \group_begin:
8230     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8231     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8232     \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8233     \@@_cut_on_hyphen:w #2 \q_stop
8234     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8235     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8236     \@@_cut_on_hyphen:w #3 \q_stop
8237     \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8238     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8239     \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8240     {
8241         \use:e
8242         {
8243             \@@_hline:n
8244             {
8245                 position = ##1 ,
8246                 start = \l_@@_tmpd_tl ,
8247                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
8248                 total-width = \dim_use:N \l_@@_line_width_dim
8249             }
8250         }
8251     }
8252     \group_end:
8253 }

```

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).

```

8254 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8255 {
8256     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8257     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8258     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8259     { \@@_error:n { borders~forbidden } }
8260     {
8261         \tl_clear_new:N \l_@@_borders_tikz_tl
8262         \keys_set:no
8263         { nicematrix / OnlyForTikzInBorders }
8264         \l_@@_borders_clist
8265         \@@_cut_on_hyphen:w #2 \q_stop
8266         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8267         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8268         \@@_cut_on_hyphen:w #3 \q_stop
8269         \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8270         \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8271         \@@_stroke_borders_block_i:
8272     }
8273 }
8274 \hook_gput_code:nnn { begindocument } { . }

```



```

8275 {
8276   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8277   {
8278     \c_@@_pgfortikzpicture_tl
8279     \@@_stroke_borders_block_ii:
8280     \c_@@_endpgfortikzpicture_tl
8281   }
8282 }

8283 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8284 {
8285   \pgfrememberpicturepositiononpagetrue
8286   \pgf@relevantforpicturesizefalse
8287   \CT@arc@
8288   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8289   \clist_if_in:NnT \l_@@_borders_clist { right }
8290   { \@@_stroke_vertical:n \l_tmpb_tl }
8291   \clist_if_in:NnT \l_@@_borders_clist { left }
8292   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8293   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8294   { \@@_stroke_horizontal:n \l_tmpa_tl }
8295   \clist_if_in:NnT \l_@@_borders_clist { top }
8296   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8297 }

8298 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8299 {
8300   tikz .code:n =
8301     \cs_if_exist:NTF \tikzpicture
8302     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8303     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8304   tikz .value_required:n = true ,
8305   top .code:n = ,
8306   bottom .code:n = ,
8307   left .code:n = ,
8308   right .code:n = ,
8309   unknown .code:n = \@@_error:n { bad~border }
8310 }

```

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).

```

8311 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8312 {
8313   \@@_qpoint:n \l_@@_tmpc_tl
8314   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8315   \@@_qpoint:n \l_tmpa_tl
8316   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8317   \@@_qpoint:n { #1 }
8318   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8319   {
8320     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8321     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8322     \pgfusepathqstroke
8323   }
8324   {
8325     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8326     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8327   }
8328 }

```

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).

```

8329 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8330 {

```

```

8331 \@@_qpoint:n \l_@@_tmpd_tl
8332 \clist_if_in:NnTF \l_@@_borders_clist { left }
8333   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8334   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8335 \@@_qpoint:n \l_tmpb_tl
8336 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8337 \@@_qpoint:n { #1 }
8338 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8339   {
8340     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8341     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8342     \pgfusepathqstroke
8343   }
8344   {
8345     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8346     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8347   }
8348 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8349 \keys_define:nn { nicematrix / BlockBorders }
8350 {
8351   borders .clist_set:N = \l_@@_borders_clist ,
8352   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8353   rounded-corners .default:n = 4 pt ,
8354   line-width .dim_set:N = \l_@@_line_width_dim
8355 }

```

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.

```

8356 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8357 {
8358   \begin { tikzpicture }
8359   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8360 \clist_map_inline:nn { #1 }
8361 {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8362 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8363 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8364 (
8365   [
8366     xshift = \dim_use:N \l_@@_offset_dim ,
8367     yshift = - \dim_use:N \l_@@_offset_dim
8368   ]
8369   #2 -| #3
8370 )
8371 rectangle
8372 (
8373   [
8374     xshift = - \dim_use:N \l_@@_offset_dim ,
8375     yshift = \dim_use:N \l_@@_offset_dim
8376   ]
8377   \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8378 ) ;

```

```

8379     }
8380   \end { tikzpicture }
8381 }
8382 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8383 \keys_define:nn { nicematrix / SpecialOffset }
8384 { 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.

```

8385 \cs_new_protected:Npn \@@_NullBlock:
8386 { \@@_collect_options:n { \@@_NullBlock_i: } }
8387 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8388 { }

```

## 27 How to draw the dotted lines transparently

```

8389 \cs_set_protected:Npn \@@_renew_matrix:
8390 {
8391   \RenewDocumentEnvironment { pmatrix } { } {
8392     { \pNiceMatrix }
8393     { \endpNiceMatrix }
8394   \RenewDocumentEnvironment { vmatrix } { } {
8395     { \vNiceMatrix }
8396     { \endvNiceMatrix }
8397   \RenewDocumentEnvironment { Vmatrix } { } {
8398     { \VNiceMatrix }
8399     { \endVNiceMatrix }
8400   \RenewDocumentEnvironment { bmatrix } { } {
8401     { \bNiceMatrix }
8402     { \endbNiceMatrix }
8403   \RenewDocumentEnvironment { Bmatrix } { } {
8404     { \BNiceMatrix }
8405     { \endBNiceMatrix }
8406   }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8407 \keys_define:nn { nicematrix / Auto }
8408 {
8409   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8410   columns-type .value_required:n = true ,
8411   l .meta:n = { columns-type = l } ,
8412   r .meta:n = { columns-type = r } ,
8413   c .meta:n = { columns-type = c } ,
8414   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8415   delimiters / color .value_required:n = true ,
8416   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8417   delimiters / max-width .default:n = true ,
8418   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8419   delimiters .value_required:n = true ,
8420   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8421   rounded-corners .default:n = 4 pt
8422 }

```

```

8423 \NewDocumentCommand \AutoNiceMatrixWithDelims
8424 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8425 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8426 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8427 {

```

The group is for the protection of the keys.

```

8428 \group_begin:
8429 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8430 \use:e
8431 {
8432 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8433 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8434 [ \exp_not:o \l_tmpa_tl ]
8435 }
8436 \int_if_zero:nT { \l_@@_first_row_int }
8437 {
8438 \int_if_zero:nT { \l_@@_first_col_int } { & }
8439 \prg_replicate:nn { #4 - 1 } { & }
8440 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8441 }
8442 \prg_replicate:nn { #3 }
8443 {
8444 \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).

```

8445 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8446 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8447 }
8448 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8449 {
8450 \int_if_zero:nT { \l_@@_first_col_int } { & }
8451 \prg_replicate:nn { #4 - 1 } { & }
8452 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8453 }
8454 \end { NiceArrayWithDelims }
8455 \group_end:
8456 }

8457 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8458 {
8459 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8460 {
8461 \bool_gset_true:N \g_@@_delims_bool
8462 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8463 \AutoNiceMatrixWithDelims { #2 } { #3 }
8464 }
8465 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8466 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8467 {
8468 \group_begin:
8469 \bool_gset_false:N \g_@@_delims_bool
8470 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8471 \group_end:
8472 }

```

## 29 The redefinition of the command `\dotfill`

```

8473 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8474 \cs_new_protected:Npn \@@_dotfill:
8475 {

```

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}`).

```

8476 \@@_old_dotfill:
8477 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8478 }

```

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`.

```

8479 \cs_new_protected:Npn \@@_dotfill_i:
8480 {
8481   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8482     { \@@_old_dotfill: }
8483 }

```

## 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.

```

8484 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8485 {
8486   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8487   {
8488     \@@_actually_diagbox:nnnnnn
8489     { \int_use:N \c@iRow }
8490     { \int_use:N \c@jCol }
8491     { \int_use:N \c@iRow }
8492     { \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.

```

8493   { \g_@@_row_style_tl \exp_not:n { #1 } }
8494   { \g_@@_row_style_tl \exp_not:n { #2 } }
8495 }

```

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`.

```

8496 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8497 {
8498   { \int_use:N \c@iRow }
8499   { \int_use:N \c@jCol }
8500   { \int_use:N \c@iRow }
8501   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8502   { }
8503 }
8504 }

```

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.

```

8505 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8506 {
8507   \pgfpicture
8508   \pgf@relevantforpicturesizefalse
8509   \pgfrememberpicturepositiononpagetrue
8510   \@@_qpoint:n { row - #1 }
8511   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8512   \@@_qpoint:n { col - #2 }
8513   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8514   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8515   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8516   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8517   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8518   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8519   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8520 }
```

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.

```

8521   \CT@arc@
8522   \pgfsetroundcap
8523   \pgfusepathqstroke
8524 }
8525 \pgfset { inner~sep = 1 pt }
8526 \pgfscope
8527 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8528 \pgfnode { rectangle } { south~west }
8529 {
8530   \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8531   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8532   \end { minipage }
8533 }
8534 { }
8535 { }
8536 \endpgfscope
8537 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8538 \pgfnode { rectangle } { north~east }
8539 {
8540   \begin { minipage } { 20 cm }
8541   \raggedleft
8542   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8543   \end { minipage }
8544 }
8545 { }
8546 { }
8547 \endpgfpicture
8548 }
```

## 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. 86.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8549 \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 `\`.

```
8550 \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`.

```
8551 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8552 {
8553   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8554   \@@_CodeAfter_iv:n
8555 }
```

We catch the argument of the command `\end` (in `#1`).

```
8556 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8557 {
```

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.

```
8558   \str_if_eq:eeTF { \currenvir } { #1 }
8559   { \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`.

```
8560   {
8561     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8562     \@@_CodeAfter_ii:n
8563   }
8564 }
```

## 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.

```
8565 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8566 {
8567   \pgfpicture
8568   \pgfrememberpicturepositiononpagetrue
8569   \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.

```
8570   \@@_qpoint:n { row - 1 }
8571   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8572   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8573   \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).

```

8574 \bool_if:nTF { #3 }
8575 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8576 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8577 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8578 {
8579   \cs_if_exist:cT
8580   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8581   {
8582     \pgfpointanchor
8583     { \@@_env: - ##1 - #2 }
8584     { \bool_if:nTF { #3 } { west } { east } }
8585     \dim_set:Nn \l_tmpa_dim
8586     {
8587       \bool_if:nTF { #3 }
8588       { \dim_min:nn }
8589       { \dim_max:nn }
8590       \l_tmpa_dim
8591       { \pgf@x }
8592     }
8593   }
8594 }

```

Now we can put the delimiter with a node of PGF.

```

8595 \pgfset { inner~sep = \c_zero_dim }
8596 \dim_zero:N \nulldelimiterspace
8597 \pgftransformshift
8598 {
8599   \pgfpoint
8600   { \l_tmpa_dim }
8601   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8602 }
8603 \pgfnode
8604 { rectangle }
8605 { \bool_if:nTF { #3 } { east } { west } }
8606 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8607 \nullfont
8608 \c_math_toggle_token
8609 \@@_color:o \l_@@_delimiters_color_tl
8610 \bool_if:nTF { #3 } { \left #1 } { \left . }
8611 \vcenter
8612 {
8613   \nullfont
8614   \hrule \@@height
8615   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8616   \@@depth \c_zero_dim
8617   \@@width \c_zero_dim
8618 }
8619 \bool_if:nTF { #3 } { \right . } { \right #1 }
8620 \c_math_toggle_token
8621 }
8622 { }
8623 { }
8624 \endpgfpicture
8625 }

```

### 33 The command `\SubMatrix`



```

8626 \keys_define:nn { nicematrix / sub-matrix }
8627 {
8628   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8629   extra-height .value_required:n = true ,
8630   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8631   left-xshift .value_required:n = true ,
8632   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8633   right-xshift .value_required:n = true ,
8634   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8635   xshift .value_required:n = true ,
8636   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8637   delimiters / color .value_required:n = true ,
8638   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8639   slim .default:n = true ,
8640   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8641   hlines .default:n = all ,
8642   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8643   vlines .default:n = all ,
8644   hvlines .meta:n = { hlines, vlines } ,
8645   hvlines .value_forbidden:n = true
8646 }
8647 \keys_define:nn { nicematrix }
8648 {
8649   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8650   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8651   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8652   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8653 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8654 \keys_define:nn { nicematrix / SubMatrix }
8655 {
8656   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8657   delimiters / color .value_required:n = true ,
8658   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8659   hlines .default:n = all ,
8660   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8661   vlines .default:n = all ,
8662   hvlines .meta:n = { hlines, vlines } ,
8663   hvlines .value_forbidden:n = true ,
8664   name .code:n =
8665     \tl_if_empty:nTF { #1 }
8666     { \@@_error:n { Invalid-name } }
8667     {
8668       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8669       {
8670         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8671         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8672         {
8673           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8674           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8675         }
8676       }
8677       { \@@_error:n { Invalid-name } }
8678     } ,
8679   name .value_required:n = true ,
8680   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8681   rules .value_required:n = true ,
8682   code .tl_set:N = \l_@@_code_tl ,
8683   code .value_required:n = true ,
8684   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8685 }

```

```

8686 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8687 {
8688   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8689   {
8690     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8691     [
8692       delimiters / color = \l_@@_delimiters_color_tl ,
8693       hlines = \l_@@_submatrix_hlines_clist ,
8694       vlines = \l_@@_submatrix_vlines_clist ,
8695       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8696       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8697       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8698       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8699       #5
8700     ]
8701   }
8702   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8703   \ignorespaces
8704 }
8705 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8706 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8707 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8708 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8709 {
8710   \seq_gput_right:Ne \g_@@_submatrix_seq
8711   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8712   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8713   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8714   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8715   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8716   }
8717 }

```

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).

```

8718 \NewDocumentCommand \@@_compute_i_j:nn
8719 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8720 { \@@_compute_i_j:nnnn #1 #2 }
8721 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8722 {
8723   \def \l_@@_first_i_tl { #1 }
8724   \def \l_@@_first_j_tl { #2 }
8725   \def \l_@@_last_i_tl { #3 }
8726   \def \l_@@_last_j_tl { #4 }
8727   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8728   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8729   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8730   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8731   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8732   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8733   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8734   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8735 }

```

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`.

```

8736 \hook_gput_code:nnn { begindocument } { . }
8737 {
8738   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8739   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8740     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8741 }
8742 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8743 {
8744   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8745   \@@_compute_i_j:nn { #2 } { #3 }
8746   \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8747   { \def \arraystretch { 1 } }
8748   \bool_lazy_or:nnTF
8749     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8750     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8751     { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8752   {
8753     \str_clear_new:N \l_@@_submatrix_name_str
8754     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8755     \pgfpicture
8756     \pgfrememberpicturepositiononpagetrue
8757     \pgf@relevantforpicturesizefalse
8758     \pgfset { inner~sep = \c_zero_dim }
8759     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8760     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

8761   \bool_if:NTF \l_@@_submatrix_slim_bool
8762   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8763   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8764   {
8765     \cs_if_exist:cT
8766     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8767     {
8768       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8769       \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8770       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8771     }
8772     \cs_if_exist:cT
8773     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8774     {
8775       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8776       \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8777       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8778     }
8779   }
8780   \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8781   { \@@_error:nn { Impossible~delimiter } { left } }
8782   {
8783     \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }

```

```

8784         { \@@_error:nn { Impossible-delimiter } { right } }
8785         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8786     }
8787     \endpgfpicture
8788 }
8789 \group_end:
8790 \ignorespaces
8791 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8792 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8793 {
8794     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8795     \dim_set:Nn \l_@@_y_initial_dim
8796     {
8797         \fp_to_dim:n
8798         {
8799             \pgf@y
8800             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8801         }
8802     }
8803     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8804     \dim_set:Nn \l_@@_y_final_dim
8805     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8806     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8807     {
8808         \cs_if_exist:cT
8809         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8810         {
8811             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8812             \dim_set:Nn \l_@@_y_initial_dim
8813             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8814         }
8815         \cs_if_exist:cT
8816         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8817         {
8818             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8819             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8820             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8821         }
8822     }
8823     \dim_set:Nn \l_tmpa_dim
8824     {
8825         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8826         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8827     }
8828     \dim_zero:N \nullldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8829     \group_begin:
8830     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8831     \@@_set_CTarc:o \l_@@_rules_color_tl
8832     \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`.

```

8833     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8834     {
8835         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8836         {
8837             \int_compare:nNnT
8838             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```

8839         {
First, we extract the value of the abscissa of the rule we have to draw.
8840         \@_qpoint:n { col - ##1 }
8841         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8842         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8843         \pgfusepathqstroke
8844     }
8845 }
8846 }

```

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.

```

8847 \str_if_eq:eeTF { \l_@_submatrix_vlines_clist } { all }
8848 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
8849 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
8850 {
8851     \bool_lazy_and:nnTF
8852     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8853     {
8854         \int_compare_p:nNn
8855         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
8856     {
8857         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
8858         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
8859         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
8860         \pgfusepathqstroke
8861     }
8862     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8863 }

```

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.

```

8864 \str_if_eq:eeTF { \l_@_submatrix_hlines_clist } { all }
8865 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
8866 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
8867 {
8868     \bool_lazy_and:nnTF
8869     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8870     {
8871         \int_compare_p:nNn
8872         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
8873     {
8874         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8875 \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8876 \dim_set:Nn \l_tmpa_dim
8877 { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
8878 \str_case:nn { #1 }
8879 {
8880     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8881     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8882     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8883     }
8884     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8885 \dim_set:Nn \l_tmpb_dim
8886 { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
8887 \str_case:nn { #2 }
8888 {
8889     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }

```

```

8890         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8891         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8892     }
8893     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8894     \pgfusepathqstroke
8895     \group_end:
8896 }
8897 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8898 }

```

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).

```

8899 \str_if_empty:NF \l_@@_submatrix_name_str
8900 {
8901     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8902     \l_@@_x_initial_dim \l_@@_y_initial_dim
8903     \l_@@_x_final_dim \l_@@_y_final_dim
8904 }
8905 \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`.

```

8906 \begin { pgfscope }
8907 \pgftransformshift
8908 {
8909     \pgfpoint
8910     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8911     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8912 }
8913 \str_if_empty:NTF \l_@@_submatrix_name_str
8914 { \@@_node_left:nn #1 { } }
8915 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8916 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8917 \pgftransformshift
8918 {
8919     \pgfpoint
8920     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8921     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8922 }
8923 \str_if_empty:NTF \l_@@_submatrix_name_str
8924 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8925 {
8926     \@@_node_right:nnnn #2
8927     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8928 }

```

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`.

```

8929 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8930 \flag_clear_new:N \l_@@_code_flag
8931 \l_@@_code_tl
8932 }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the  $i$  and  $j$  in specifications of nodes of the forms  $i-j$ ,  $\text{row-}i$ ,  $\text{col-}j$  and  $i-|j$  refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8933 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
8934 \cs_new:Npn \@@_pgfpointanchor:n #1
8935 { \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`).

```
8936 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8937 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

8938 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8939 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8940 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8941 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8942 { \@@_pgfpointanchor_ii:n { #1 } }
8943 }
```

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`).

```
8944 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8945 { \@@_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` 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`.

```
8946 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

8947 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8948 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8949 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8950 { \@@_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`).

```
8951 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8952 }
```

The following function is for the case when the name contains an hyphen.

```
8953 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8954 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8955 \@@_env:
8956 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8957 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8958 }
```

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.

```

8959 \tl_const:Nn \c_@@_integers_alist_tl
8960 {
8961   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8962   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8963   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8964   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8965 }

8966 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8967 {

```

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`.

```

8968   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8969   {
8970     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8971   \@@_env: -
8972   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8973     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8974     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8975   }
8976   {
8977     \str_if_eq:eeTF { #1 } { last }
8978     {
8979       \flag_raise:N \l_@@_code_flag
8980       \@@_env: -
8981       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8982         { \int_eval:n { \l_@@_last_i_tl + 1 } }
8983         { \int_eval:n { \l_@@_last_j_tl + 1 } }
8984     }
8985     { #1 }
8986   }
8987 }

```

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`).

```

8988 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8989 {
8990   \pgfnode
8991     { rectangle }
8992     { east }
8993     {
8994       \nullfont
8995       \c_math_toggle_token
8996       \@@_color:o \l_@@_delimiters_color_tl
8997       \left #1
8998       \vcenter
8999       {
9000         \nullfont
9001         \hrule \@height \l_tmpa_dim
9002         \@depth \c_zero_dim

```



```

9003             \@width \c_zero_dim
9004         }
9005         \right .
9006         \c_math_toggle_token
9007     }
9008     { #2 }
9009     { }
9010 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```

9011 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9012 {
9013     \pgfnode
9014     { rectangle }
9015     { west }
9016     {
9017         \nullfont
9018         \c_math_toggle_token
9019         \colorlet { current-color } { . }
9020         \@@_color:o \l_@@_delimiters_color_tl
9021         \left .
9022         \vcenter
9023         {
9024             \nullfont
9025             \hrule \@height \l_tmpa_dim
9026                 \@depth \c_zero_dim
9027                 \@width \c_zero_dim
9028         }
9029         \right #1
9030         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9031         ^ { \color { current-color } \smash { #4 } }
9032         \c_math_toggle_token
9033     }
9034     { #2 }
9035     { }
9036 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9037 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9038 {
9039     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9040     \ignorespaces
9041 }
9042 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9043 {
9044     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9045     \ignorespaces
9046 }
9047 \keys_define:nn { nicematrix / Brace }
9048 {
9049     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9050     left-shorten .default:n = true ,
9051     left-shorten .value_forbidden:n = true ,

```

```

9052 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9053 right-shorten .default:n = true ,
9054 right-shorten .value_forbidden:n = true ,
9055 shorten .meta:n = { left-shorten , right-shorten } ,
9056 shorten .value_forbidden:n = true ,
9057 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9058 yshift .value_required:n = true ,
9059 yshift .initial:n = \c_zero_dim ,
9060 color .tl_set:N = \l_tmpa_tl ,
9061 color .value_required:n = true ,
9062 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9063 }

```

#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.

```

9064 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9065 {
9066   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9067   \@@_compute_i_j:nn { #1 } { #2 }
9068   \bool_lazy_or:nnTF
9069     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9070     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9071     {
9072       \str_if_eq:eeTF { #5 } { under }
9073       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9074       { \@@_error:nn { Construct~too~large } { \OverBrace } }
9075     }
9076     {
9077       \tl_clear:N \l_tmpa_tl
9078       \keys_set:nn { nicematrix / Brace } { #4 }
9079       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9080       \pgfpicture
9081       \pgfrememberpicturepositiononpagetrue
9082       \pgf@relevantforpicturesizefalse
9083       \bool_if:NT \l_@@_brace_left_shorten_bool
9084       {
9085         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9086         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9087         {
9088           \cs_if_exist:cT
9089             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9090             {
9091               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9092
9093               \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9094               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9095             }
9096         }
9097       }
9098       \bool_lazy_or:nnT
9099         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9100         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9101         {
9102           \@@_qpoint:n { col - \l_@@_first_j_tl }
9103           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9104         }
9105       \bool_if:NT \l_@@_brace_right_shorten_bool
9106       {
9107         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9108         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9109         {

```

```

9110         \cs_if_exist:cT
9111         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9112         {
9113             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9114             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9115             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9116         }
9117     }
9118 }
9119 \bool_lazy_or:nnT
9120 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9121 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9122 {
9123     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9124     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9125 }
9126 \pgfset { inner~sep = \c_zero_dim }
9127 \str_if_eq:eeTF { #5 } { under }
9128 { \@@_underbrace_i:n { #3 } }
9129 { \@@_overbrace_i:n { #3 } }
9130 \endpgfpicture
9131 }
9132 \group_end:
9133 }

```

The argument is the text to put above the brace.

```

9134 \cs_new_protected:Npn \@@_overbrace_i:n #1
9135 {
9136     \@@_qpoint:n { row - \l_@@_first_i_tl }
9137     \pgftransformshift
9138     {
9139         \pgfpoint
9140         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9141         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9142     }
9143     \pgfnode
9144     { rectangle }
9145     { south }
9146     {
9147         \vtop
9148         {
9149             \group_begin:
9150             \everycr { }
9151             \halign
9152             {
9153                 \hfil ## \hfil \crcr
9154                 \bool_if:NTF \l_@@_tabular_bool
9155                 { \begin { tabular } { c } #1 \end { tabular } }
9156                 { $ \begin { array } { c } #1 \end { array } $ }
9157                 \cr
9158                 \c_math_toggle_token
9159                 \overbrace
9160                 {
9161                     \hbox_to_wd:nn
9162                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9163                     { }
9164                 }
9165                 \c_math_toggle_token
9166                 \cr
9167             }
9168             \group_end:
9169         }
9170     }
9171     { }

```

```

9172     { }
9173 }

```

The argument is the text to put under the brace.

```

9174 \cs_new_protected:Npn \@@_underbrace_i:n #1
9175 {
9176   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9177   \pgftransformshift
9178   {
9179     \pgfpoint
9180     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9181     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9182   }
9183   \pgfnode
9184   { rectangle }
9185   { north }
9186   {
9187     \group_begin:
9188     \everycr { }
9189     \vbox
9190     {
9191       \halign
9192       {
9193         \hfil ## \hfil \crcr
9194         \c_math_toggle_token
9195         \underbrace
9196         {
9197           \hbox_to_wd:nn
9198           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9199           { }
9200         }
9201         \c_math_toggle_token
9202         \cr
9203         \bool_if:NTF \l_@@_tabular_bool
9204         { \begin { tabular } { c } #1 \end { tabular } }
9205         { $ \begin { array } { c } #1 \end { array } $ }
9206         \cr
9207       }
9208     }
9209     \group_end:
9210   }
9211   { }
9212   { }
9213 }

```

## 35 The commands HBrace et VBrace

```

9214 \hook_gput_code:nnn { begindocument } { . }
9215 {
9216   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9217   {
9218     \tikzset
9219     {
9220       nicematrix / brace / .style =
9221       {
9222         decoration = { brace , raise = -0.15 em } ,
9223         decorate ,
9224       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9225         nicematrix / mirrored-brace / .style =
9226         {
9227             nicematrix / brace ,
9228             decoration = mirror ,
9229         }
9230     }
9231 }
9232 }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9233 \keys_define:nn { nicematrix / Hbrace }
9234 {
9235     color .code:n = ,
9236     horizontal-label .code:n = ,
9237     horizontal-labels .code:n = ,
9238     shorten .code:n = ,
9239     shorten-start .code:n = ,
9240     shorten-end .code:n = ,
9241     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9242 }
```

Here we need an “fully expandable” command.

```

9243 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9244 {
9245     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9246     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9247     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9248 }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9249 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9250 {
9251     \int_compare:nNnTF { \c@iRow } < { 2 }
9252     {
```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9253     \str_if_eq:nnTF { #2 } { * }
9254     {
9255         \NiceMatrixOptions { nullify-dots }
9256         \Ldots
9257         [
9258             line-style = nicematrix / brace ,
9259             #1 ,
9260             up =
9261             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9262         ]
9263     }
9264     {
9265         \Hdotsfor
9266         [
9267             line-style = nicematrix / brace ,
9268             #1 ,
9269             up =
9270             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9271         ]
9272         { #2 }
9273     }
9274 }
9275 {
9276     \str_if_eq:nnTF { #2 } { * }
```

```

9277 {
9278   \NiceMatrixOptions { nullify-dots }
9279   \Ldots
9280   [
9281     line-style = nicematrix / mirrored-brace ,
9282     #1 ,
9283     down =
9284       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9285   ]
9286 }
9287 {
9288   \Hdotsfor
9289   [
9290     line-style = nicematrix / mirrored-brace ,
9291     #1 ,
9292     down =
9293       \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9294   ]
9295   { #2 }
9296 }
9297 }
9298 \keys_set:nn { nicematrix / Hbrace } { #1 }
9299 }

9300 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9301 {
9302   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9303     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9304     { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9305 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9306 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9307 {
9308   \int_compare:nNnTF { \c@jCol } < { 2 }
9309   {
9310     \str_if_eq:nnTF { #2 } { * }
9311     {
9312       \NiceMatrixOptions { nullify-dots }
9313       \Vdots
9314       [
9315         Vbrace ,
9316         line-style = nicematrix / mirrored-brace ,
9317         #1 ,
9318         down =
9319           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9320       ]
9321     }
9322     {
9323       \Vdotsfor
9324       [
9325         Vbrace ,
9326         line-style = nicematrix / mirrored-brace ,
9327         #1 ,
9328         down =
9329           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9330       ]
9331       { #2 }
9332     }
9333   }
9334   {
9335     \str_if_eq:nnTF { #2 } { * }
9336     {

```

```

9337     \NiceMatrixOptions { nullify-dots }
9338     \Vdots
9339     [
9340         Vbrace ,
9341         line-style = nicematrix / brace ,
9342         #1 ,
9343         up =
9344         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9345     ]
9346 }
9347 {
9348     \Vdotsfor
9349     [
9350         Vbrace ,
9351         line-style = nicematrix / brace ,
9352         #1 ,
9353         up =
9354         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9355     ]
9356     { #2 }
9357 }
9358 }
9359 \keys_set:nn { nicematrix / Hbrace } { #1 }
9360 }

```

## 36 The command TikzEveryCell

```

9361 \bool_new:N \l_@@_not_empty_bool
9362 \bool_new:N \l_@@_empty_bool
9363
9364 \keys_define:nn { nicematrix / TikzEveryCell }
9365 {
9366     not-empty .code:n =
9367     \bool_lazy_or:nnTF
9368     { \l_@@_in_code_after_bool }
9369     { \g_@@_create_cell_nodes_bool }
9370     { \bool_set_true:N \l_@@_not_empty_bool }
9371     { \@@_error:n { detection-of-empty-cells } } ,
9372     not-empty .value_forbidden:n = true ,
9373     empty .code:n =
9374     \bool_lazy_or:nnTF
9375     { \l_@@_in_code_after_bool }
9376     { \g_@@_create_cell_nodes_bool }
9377     { \bool_set_true:N \l_@@_empty_bool }
9378     { \@@_error:n { detection-of-empty-cells } } ,
9379     empty .value_forbidden:n = true ,
9380     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9381 }
9382
9383
9384 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9385 {
9386     \IfPackageLoadedTF { tikz }
9387     {
9388         \group_begin:
9389         \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.

```

9390     \tl_set:Nn \l_tmpa_tl { { #2 } }
9391     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9392     { \@@_for_a_block:nnnnn ##1 }

```

```

9393     \@@_all_the_cells:
9394     \group_end:
9395 }
9396 { \@@_error:n { TikzEveryCell~without~tikz } }
9397 }
9398
9399
9400 \cs_new_protected:Nn \@@_all_the_cells:
9401 {
9402   \int_step_inline:nn \c@iRow
9403   {
9404     \int_step_inline:nn \c@jCol
9405     {
9406       \cs_if_exist:cF { cell - ##1 - #####1 }
9407       {
9408         \clist_if_in:NcF \l_@@_corners_cells_clist
9409         { ##1 - #####1 }
9410         {
9411           \bool_set_false:N \l_tmpa_bool
9412           \cs_if_exist:cTF
9413           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9414           {
9415             \bool_if:NF \l_@@_empty_bool
9416             { \bool_set_true:N \l_tmpa_bool }
9417           }
9418           {
9419             \bool_if:NF \l_@@_not_empty_bool
9420             { \bool_set_true:N \l_tmpa_bool }
9421           }
9422           \bool_if:NT \l_tmpa_bool
9423           {
9424             \@@_block_tikz:nnnnn
9425             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9426           }
9427         }
9428       }
9429     }
9430   }
9431 }
9432
9433 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9434 {
9435   \bool_if:NF \l_@@_empty_bool
9436   {
9437     \@@_block_tikz:nnnnn
9438     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9439   }
9440   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9441 }
9442
9443 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9444 {
9445   \int_step_inline:nnn { #1 } { #3 }
9446   {
9447     \int_step_inline:nnn { #2 } { #4 }
9448     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9449   }
9450 }

```

## 37 The command \ShowCellNames

```

9451 \NewDocumentCommand \@@_ShowCellNames { }

```



```

9452 {
9453   \bool_if:NT \l_@@_in_code_after_bool
9454   {
9455     \pgfpicture
9456     \pgfrememberpicturepositiononpagetrue
9457     \pgf@relevantforpicturesizefalse
9458     \pgfpathrectanglecorners
9459     { \@@_qpoint:n { 1 } }
9460     {
9461       \@@_qpoint:n
9462       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9463     }
9464     \pgfsetfillopacity { 0.75 }
9465     \pgfsetfillcolor { white }
9466     \pgfusepathqfill
9467     \endpgfpicture
9468   }
9469   \dim_gzero_new:N \g_@@_tmpc_dim
9470   \dim_gzero_new:N \g_@@_tmpd_dim
9471   \dim_gzero_new:N \g_@@_tmpe_dim
9472   \int_step_inline:nn { \c@iRow }
9473   {
9474     \bool_if:NTF \l_@@_in_code_after_bool
9475     {
9476       \pgfpicture
9477       \pgfrememberpicturepositiononpagetrue
9478       \pgf@relevantforpicturesizefalse
9479     }
9480     { \begin { pgfpicture } }
9481     \@@_qpoint:n { row - ##1 }
9482     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9483     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9484     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9485     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9486     \bool_if:NTF \l_@@_in_code_after_bool
9487     { \endpgfpicture }
9488     { \end { pgfpicture } }
9489     \int_step_inline:nn { \c@jCol }
9490     {
9491       \hbox_set:Nn \l_tmpa_box
9492       {
9493         \normalfont \Large \sffamily \bfseries
9494         \bool_if:NTF \l_@@_in_code_after_bool
9495         { \color { red } }
9496         { \color { red ! 50 } }
9497         ##1 - ####1
9498       }
9499       \bool_if:NTF \l_@@_in_code_after_bool
9500       {
9501         \pgfpicture
9502         \pgfrememberpicturepositiononpagetrue
9503         \pgf@relevantforpicturesizefalse
9504       }
9505       { \begin { pgfpicture } }
9506       \@@_qpoint:n { col - ####1 }
9507       \dim_set_eq:NN \g_@@_tmpc_dim \pgf@x
9508       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9509       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9510       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9511       \bool_if:NTF \l_@@_in_code_after_bool
9512       { \endpgfpicture }
9513       { \end { pgfpicture } }
9514       \fp_set:Nn \l_tmpa_fp

```

```

9515         {
9516             \fp_min:nn
9517             {
9518                 \fp_min:nn
9519                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9520                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9521             }
9522             { 1.0 }
9523         }
9524         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9525         \pgfpicture
9526         \pgfrememberpicturepositiononpagetrue
9527         \pgf@relevantforpicturesizefalse
9528         \pgftransformshift
9529         {
9530             \pgfpoint
9531             { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9532             { \dim_use:N \g_tmpa_dim }
9533         }
9534         \pgfnode
9535         { rectangle }
9536         { center }
9537         { \box_use:N \l_tmpa_box }
9538         { }
9539         { }
9540         \endpgfpicture
9541     }
9542 }
9543 }

```

## 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.

```

9544 \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.

```

9545 \bool_new:N \g_@@_footnote_bool

9546 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9547 {
9548     You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9549     but-that-key-is-unknown. \\
9550     It-will-be-ignored. \\
9551     For-a-list-of-the-available-keys,~type-H~<return>.
9552 }
9553 {
9554     The-available-keys-are~(in~alphabetic~order):~
9555     footnote,~
9556     footnotehyper,~
9557     messages-for-Overleaf,~
9558     renew-dots-and~
9559     renew-matrix.
9560 }

```

```

9561 \keys_define:nn { nicematrix }
9562 {
9563   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9564   renew-dots .value_forbidden:n = true ,
9565   renew-matrix .code:n = \@@_renew_matrix: ,
9566   renew-matrix .value_forbidden:n = true ,
9567   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9568   footnote .bool_set:N = \g_@@_footnote_bool ,
9569   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9570   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9571 }
9572 \ProcessKeyOptions

9573 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9574 {
9575   You~can't~use~the~option~'footnote'~because~the~package~
9576   footnotehyper~has~already~been~loaded.~
9577   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9578   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9579   of~the~package~footnotehyper.\\
9580   The~package~footnote~won't~be~loaded.
9581 }

9582 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9583 {
9584   You~can't~use~the~option~'footnotehyper'~because~the~package~
9585   footnote~has~already~been~loaded.~
9586   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9587   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9588   of~the~package~footnote.\\
9589   The~package~footnotehyper~won't~be~loaded.
9590 }

9591 \bool_if:NT \g_@@_footnote_bool
9592 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9593 \IfClassLoadedTF { beamer }
9594 { \bool_set_false:N \g_@@_footnote_bool }
9595 {
9596   \IfPackageLoadedTF { footnotehyper }
9597   { \@@_error:n { footnote~with~footnotehyper~package } }
9598   { \usepackage { footnote } }
9599 }
9600 }

9601 \bool_if:NT \g_@@_footnotehyper_bool
9602 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9603 \IfClassLoadedTF { beamer }
9604 { \bool_set_false:N \g_@@_footnote_bool }
9605 {
9606   \IfPackageLoadedTF { footnote }
9607   { \@@_error:n { footnotehyper~with~footnote~package } }
9608   { \usepackage { footnotehyper } }
9609 }
9610 \bool_set_true:N \g_@@_footnote_bool
9611 }

```

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.

```
9612 \bool_new:N \l_@@_underscore_loaded_bool
9613 \IfPackageLoadedT { underscore }
9614 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9615 \hook_gput_code:nnn { begindocument } { . }
9616 {
9617   \bool_if:NF \l_@@_underscore_loaded_bool
9618   {
9619     \IfPackageLoadedT { underscore }
9620     { \@@_error:n { underscore~after~nicematrix } }
9621   }
9622 }
```

## 40 Error messages of the package

```
9623 \str_const:Ne \c_@@_available_keys_str
9624 {
9625   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9626   { For~a~list~of~the~available~keys,~type~H~<return>. }
9627   { }
9628 }
9629 \seq_new:N \g_@@_types_of_matrix_seq
9630 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9631 {
9632   NiceMatrix ,
9633   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9634 }
9635 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9636 { \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`.

```
9637 \cs_new_protected:Npn \@@_error_too_much_cols:
9638 {
9639   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9640   { \@@_fatal:nn { too-much-cols-for-array } }
9641   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9642   { \@@_fatal:n { too-much-cols-for-matrix } }
9643   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9644   { \@@_fatal:n { too-much-cols-for-matrix } }
9645   \bool_if:NF \l_@@_last_col_without_value_bool
9646   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9647 }
```

The following command must *not* be protected since it's used in an error message.

```
9648 \cs_new:Npn \@@_message_hdotsfor:
9649 {
9650   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9651   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9652     \token_to_str:N \Hbrace \ is~incorrect. }
9653 }
```

```

9654 \cs_new_protected:Npn \@@_Hline_in_cell:
9655 { \@@_fatal:n { Misuse-of~Hline } }

9656 \@@_msg_new:nn { Misuse-of~Hline }
9657 {
9658   Misuse-of~Hline. \\
9659   \token_to_str:N \Hline\ must-be-used-only-at-the-beginning-of-a-row.\\
9660   That-error-is-fatal.
9661 }

9662 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9663 {
9664   Incompatible-options.\\
9665   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-the-same-time.\\
9666   The-output-will-not-be-reliable.
9667 }

9668 \@@_msg_new:nn { key-color-inside }
9669 {
9670   Key-deprecated.\\
9671   The-key-'color-inside'~(and-its-alias-'colortbl-like')~is-now-point-less~
9672   and-have-been-deprecated.\\
9673   You-won't-have-similar-message-till-the-end-of-the-document.
9674 }

9675 \@@_msg_new:nn { invalid-weight }
9676 {
9677   Unknown-key.\\
9678   The-key~'\l_keys_key_str'~of-your-column-X-is-unknown-and-will-be-ignored.
9679 }

9680 \@@_msg_new:nn { last-col-not-used }
9681 {
9682   Column-not-used.\\
9683   The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
9684   in~your~\@@_full_name_env: .~
9685   However,~you-can~go~on.
9686 }

9687 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9688 {
9689   Too-much-columns.\\
9690   In-the-row~\int_eval:n { \c@iRow },~
9691   you-try-to-use-more-columns~
9692   than-allowed-by-your~\@@_full_name_env: .
9693   \@@_message_hdotsfor: \
9694   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9695   (plus-the-exterior-columns).~This-error-is-fatal.
9696 }

9697 \@@_msg_new:nn { too-much-cols-for-matrix }
9698 {
9699   Too-much-columns.\\
9700   In-the-row~\int_eval:n { \c@iRow },~
9701   you-try-to-use-more-columns-than-allowed-by-your~\@@_full_name_env: .
9702   \@@_message_hdotsfor: \
9703   Recall-that-the-maximal-number-of-columns-for-a-matrix~
9704   (excepted-the-potential-exterior-columns)~is-fixed-by-the~
9705   LaTeX-counter~'MaxMatrixCols'.~
9706   Its-current-value-is~\int_use:N \c@MaxMatrixCols \
9707   (use~\token_to_str:N \setcounter \ to-change-that-value).~
9708   This-error-is-fatal.
9709 }

9710 \@@_msg_new:nn { too-much-cols-for-array }
9711 {
9712   Too-much-columns.\\
9713   In-the-row~\int_eval:n { \c@iRow },~

```

```

9714 ~you-try-to-use-more-columns-than-allowed-by-your~
9715 \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal-number-of-columns-is~
9716 \int_use:N \g_@@_static_num_of_col_int \
9717 \bool_if:nT
9718 { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9719 { ~(plus-the-exterior-ones) }
9720 since-the-preamble-is~' \g_@@_user_preamble_tl '.\\
9721 This-error-is-fatal.
9722 }
9723 \@@_msg_new:nn { columns~not~used }
9724 {
9725 Columns~not~used.\\
9726 The~preamble-of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
9727 It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9728 columns~but~you~only~used~ \int_use:N \c@jCol .\\
9729 The~columns~you~did~not~used~won't~be~created.\\
9730 You~won't~have~similar~warning~till~the~end~of~the~document.
9731 }
9732 \@@_msg_new:nn { empty~preamble }
9733 {
9734 Empty~preamble.\\
9735 The~preamble-of~your~ \@@_full_name_env: \ is~empty.\\
9736 This-error-is-fatal.
9737 }
9738 \@@_msg_new:nn { in~first~col }
9739 {
9740 Erroneous~use.\\
9741 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9742 That~command~will~be~ignored.
9743 }
9744 \@@_msg_new:nn { in~last~col }
9745 {
9746 Erroneous~use.\\
9747 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9748 That~command~will~be~ignored.
9749 }
9750 \@@_msg_new:nn { in~first~row }
9751 {
9752 Erroneous~use.\\
9753 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9754 That~command~will~be~ignored.
9755 }
9756 \@@_msg_new:nn { in~last~row }
9757 {
9758 Erroneous~use.\\
9759 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9760 That~command~will~be~ignored.
9761 }
9762 \@@_msg_new:nn { TopRule~without~booktabs }
9763 {
9764 Erroneous~use.\\
9765 You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9766 That~command~will~be~ignored.
9767 }
9768 \@@_msg_new:nn { TopRule~without~tikz }
9769 {
9770 Erroneous~use.\\
9771 You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9772 That~command~will~be~ignored.
9773 }

```

```

9774 \@@_msg_new:nn { caption~outside~float }
9775 {
9776   Key~caption~forbidden.\\
9777   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9778   environment~(such~as~\{table\}).~This~key~will~be~ignored.
9779 }
9780 \@@_msg_new:nn { short~caption~without~caption }
9781 {
9782   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9783   However,~your~'short~caption'~will~be~used~as~'caption'.
9784 }
9785 \@@_msg_new:nn { double~closing~delimiter }
9786 {
9787   Double~delimiter.\\
9788   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9789   delimiter.~This~delimiter~will~be~ignored.
9790 }
9791 \@@_msg_new:nn { delimiter~after~opening }
9792 {
9793   Double~delimiter.\\
9794   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9795   delimiter.~That~delimiter~will~be~ignored.
9796 }
9797 \@@_msg_new:nn { bad~option~for~line~style }
9798 {
9799   Bad~line~style.\\
9800   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9801   is~'standard'.~That~key~will~be~ignored.
9802 }
9803 \@@_msg_new:nn { corners~with~no~cell~nodes }
9804 {
9805   Incompatible~keys.\\
9806   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9807   is~in~force.\\
9808   If~you~go~on,~that~key~will~be~ignored.
9809 }
9810 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9811 {
9812   Incompatible~keys.\\
9813   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9814   is~in~force.\\
9815   If~you~go~on,~those~extra~nodes~won't~be~created.
9816 }
9817 \@@_msg_new:nn { Identical~notes~in~caption }
9818 {
9819   Identical~tabular~notes.\\
9820   You~can't~put~several~notes~with~the~same~content~in~
9821   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9822   If~you~go~on,~the~output~will~probably~be~erroneous.
9823 }
9824 \@@_msg_new:nn { tabularnote~below~the~tabular }
9825 {
9826   \token_to_str:N \tabularnote \ forbidden\\
9827   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
9828   of~your~tabular~because~the~caption~will~be~composed~below~
9829   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9830   key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\\
9831   Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9832   no~similar~error~will~raised~in~this~document.
9833 }

```

```

9834 \@@_msg_new:nn { Unknown~key~for~rules }
9835 {
9836   Unknown~key.\\
9837   There~is~only~two~keys~available~here:~width~and~color.\\
9838   Your~key~' \l_keys_key_str '~will~be~ignored.
9839 }
9840 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9841 {
9842   Unknown~key.\\
9843   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9844   keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9845   and~ \token_to_str:N \Vbrace \ are:~'color',~
9846   'horizontal-label(s)',~'shorten'~'shorten-end'~
9847   and~'shorten-start'.\\
9848   That~error~is~fatal.
9849 }
9850 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9851 {
9852   Unknown~key.\\
9853   There~is~only~two~keys~available~here:~
9854   'empty'~and~'not-empty'.\\
9855   Your~key~' \l_keys_key_str '~will~be~ignored.
9856 }
9857 \@@_msg_new:nn { Unknown~key~for~rotate }
9858 {
9859   Unknown~key.\\
9860   The~only~key~available~here~is~'c'.\\
9861   Your~key~' \l_keys_key_str '~will~be~ignored.
9862 }
9863 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9864 {
9865   Unknown~key.\\
9866   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9867   It~you~go~on,~you~will~probably~have~other~errors. \\
9868   \c_@@_available_keys_str
9869 }
9870 {
9871   The~available~keys~are~(in~alphabetic~order):~
9872   ccommand,~
9873   color,~
9874   command,~
9875   dotted,~
9876   letter,~
9877   multiplicity,~
9878   sep-color,~
9879   tikz,~and~total-width.
9880 }
9881 \@@_msg_new:nnn { Unknown~key~for~xdots }
9882 {
9883   Unknown~key.\\
9884   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9885   \c_@@_available_keys_str
9886 }
9887 {
9888   The~available~keys~are~(in~alphabetic~order):~
9889   'color',~
9890   'horizontal(s)-labels',~
9891   'inter',~
9892   'line-style',~
9893   'radius',~
9894   'shorten',~
9895   'shorten-end'~and~'shorten-start'.

```



```

9896     }
9897 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9898 {
9899     Unknown~key.\\
9900     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9901     (and~you~try~to~use~' \l_keys_key_str ')\
9902     That~key~will~be~ignored.
9903 }
9904 \@@_msg_new:nn { label~without~caption }
9905 {
9906     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9907     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9908 }
9909 \@@_msg_new:nn { W~warning }
9910 {
9911     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
9912     (row~ \int_use:N \c@iRow ).
9913 }
9914 \@@_msg_new:nn { Construct~too~large }
9915 {
9916     Construct~too~large.\\
9917     Your~command~ \token_to_str:N #1
9918     can't~be~drawn~because~your~matrix~is~too~small.\\
9919     That~command~will~be~ignored.
9920 }
9921 \@@_msg_new:nn { underscore~after~nicematrix }
9922 {
9923     Problem~with~'underscore'.\\
9924     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9925     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9926     ' \token_to_str:N \Cdots \token_to_str:N _
9927     \{ n \token_to_str:N \text \{ ~times \} \}' .
9928 }
9929 \@@_msg_new:nn { ampersand~in~light~syntax }
9930 {
9931     Ampersand~forbidden.\\
9932     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
9933     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9934 }
9935 \@@_msg_new:nn { double~backslash~in~light~syntax }
9936 {
9937     Double~backslash~forbidden.\\
9938     You~can't~use~ \token_to_str:N \\
9939     ~to~separate~rows~because~the~key~'light-syntax'~
9940     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
9941     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9942 }
9943 \@@_msg_new:nn { hlines~with~color }
9944 {
9945     Incompatible~keys.\\
9946     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9947     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
9948     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
9949     Your~key~will~be~discarded.
9950 }
9951 \@@_msg_new:nn { bad~value~for~baseline }
9952 {
9953     Bad~value~for~baseline.\\
9954     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
9955     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~

```

```

9956 \int_use:N \g_@@_row_total_int \ or-equal-to~'t',~'c'~or~'b'~or~of~
9957 the-form~'line-i'.\\
9958 A~value~of~1~will~be~used.
9959 }
9960 \@@_msg_new:nn { detection-of-empty-cells }
9961 {
9962   Problem-with~'not-empty'\\
9963   For-technical-reasons,~you~must~activate~
9964   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
9965   in-order-to-use-the-key~' \l_keys_key_str '.\\
9966   That-key-will-be-ignored.
9967 }
9968 \@@_msg_new:nn { siunitx-not-loaded }
9969 {
9970   siunitx-not-loaded\\
9971   You-can't-use-the-columns~'S'~because~'siunitx'~is~not~loaded.\\
9972   That-error-is-fatal.
9973 }
9974 \@@_msg_new:nn { Invalid-name }
9975 {
9976   Invalid-name.\\
9977   You-can't-give-the-name~' \l_keys_value_tl '~to-a~ \token_to_str:N
9978   \SubMatrix \ of~your~ \@@_full_name_env: .\\
9979   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9980   This-key-will-be-ignored.
9981 }
9982 \@@_msg_new:nn { Hbrace-not-allowed }
9983 {
9984   Command-not-allowed.\\
9985   You-can't-use-the-command~ \token_to_str:N #1
9986   because-you-have-not-loaded~
9987   \IfPackageLoadedTF { tikz }
9988     { the-TikZ-library~'decorations.pathreplacing'~Use~ }
9989     { TikZ~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
9990   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
9991   That-command-will-be-ignored.
9992 }
9993 \@@_msg_new:nn { Vbrace-not-allowed }
9994 {
9995   Command-not-allowed.\\
9996   You-can't-use-the-command~ \token_to_str:N \Vbrace \
9997   because-you-have-not-loaded-TikZ~
9998   and-the-TikZ-library~'decorations.pathreplacing'.\\
9999   Use: ~\token_to_str:N \usepackage \{tikz\}~
10000   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10001   That-command-will-be-ignored.
10002 }
10003 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
10004 {
10005   Wrong-line.\\
10006   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
10007   \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10008   number~is~not~valid.~It~will~be~ignored.
10009 }
10010 \@@_msg_new:nn { Impossible-delimiter }
10011 {
10012   Impossible-delimiter.\\
10013   It's-impossible-to-draw-the~#1~delimiter-of~your~
10014   \token_to_str:N \SubMatrix \ because-all-the-cells-are-empty~
10015   in~that~column.
10016   \bool_if:NT \l_@@_submatrix_slim_bool

```

```

10017     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
10018     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10019 }

10020 \@@_msg_new:nnn { width~without~X~columns }
10021 {
10022     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10023     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10024     That~key~will~be~ignored.
10025 }
10026 {
10027     This~message~is~the~message~'width~without~X~columns'~
10028     of~the~module~'nicematrix'.~
10029     The~experimented~users~can~disable~that~message~with~
10030     \token_to_str:N \msg_redirect_name:nnn .\\
10031 }
10032

10033 \@@_msg_new:nn { key~multiplicity~with~dotted }
10034 {
10035     Incompatible~keys. \\
10036     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10037     in~a~'custom-line'~.~They~are~incompatible. \\
10038     The~key~'multiplicity'~will~be~discarded.
10039 }

10040 \@@_msg_new:nn { empty~environment }
10041 {
10042     Empty~environment.\\
10043     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10044 }

10045 \@@_msg_new:nn { No~letter~and~no~command }
10046 {
10047     Erroneous~use.\\
10048     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10049     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10050     '~'ccommand'~(to~draw~horizontal~rules).\\
10051     However,~you~can~go~on.
10052 }

10053 \@@_msg_new:nn { Forbidden~letter }
10054 {
10055     Forbidden~letter.\\
10056     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10057     It~will~be~ignored.\\
10058     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10059 }

10060 \@@_msg_new:nn { Several~letters }
10061 {
10062     Wrong~name.\\
10063     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10064     have~used~' \l_@@_letter_str ').\\
10065     It~will~be~ignored.
10066 }

10067 \@@_msg_new:nn { Delimiter~with~small }
10068 {
10069     Delimiter~forbidden.\\
10070     You~can't~put~a~delimiter~in~the~preamble~of~your~
10071     \@@_full_name_env: \
10072     because~the~key~'small'~is~in~force.\\
10073     This~error~is~fatal.
10074 }

10075 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10076 {

```

```

10077     Unknown~cell.\\
10078     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10079     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10080     can't~be~executed~because~a~cell~doesn't~exist.\\
10081     This~command~ \token_to_str:N \line \ will~be~ignored.
10082   }
10083   \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10084   {
10085     Duplicate~name.\\
10086     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10087     in~this~ \@@_full_name_env: .\\
10088     This~key~will~be~ignored.\\
10089     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10090     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10091   }
10092   {
10093     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10094     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10095   }
10096   \@@_msg_new:nn { r~or~l~with~preamble }
10097   {
10098     Erroneous~use.\\
10099     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10100     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10101     your~ \@@_full_name_env: .\\
10102     This~key~will~be~ignored.
10103   }
10104   \@@_msg_new:nn { Hdotsfor~in~col~0 }
10105   {
10106     Erroneous~use.\\
10107     You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10108     the~array.~This~error~is~fatal.
10109   }
10110   \@@_msg_new:nn { bad~corner }
10111   {
10112     Bad~corner.\\
10113     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10114     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10115     This~specification~of~corner~will~be~ignored.
10116   }
10117   \@@_msg_new:nn { bad~border }
10118   {
10119     Bad~border.\\
10120     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10121     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10122     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10123     also~use~the~key~'tikz'
10124     \IfPackageLoadedF { tikz }
10125     { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10126     This~specification~of~border~will~be~ignored.
10127   }
10128   \@@_msg_new:nn { TikzEveryCell~without~tikz }
10129   {
10130     TikZ~not~loaded.\\
10131     You~can't~use~ \token_to_str:N \TikzEveryCell \
10132     because~you~have~not~loaded~tikz.~
10133     This~command~will~be~ignored.
10134   }
10135   \@@_msg_new:nn { tikz~key~without~tikz }
10136   {
10137     TikZ~not~loaded.\\

```

```

10138     You-can't-use-the-key~'tikz'~for-the-command~' \token_to_str:N
10139     \Block ~because-you-have-not-loaded-tikz.~
10140     This-key-will-be-ignored.
10141   }
10142   \@@_msg_new:nn { Bad-argument-for-Block }
10143   {
10144     Bad-argument.\\
10145     The-first-mandatory-argument-of~\token_to_str:N \Block\ must~
10146     be-of-the-form~'i-j'~(or-completely-empty)~and-you-have-used:~
10147     '#1'. \\
10148     If-you-go-on,~the~\token_to_str:N \Block\ will-be-mono-cell~(as-if~
10149     the-argument-was-empty).
10150   }
10151   \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10152   {
10153     Erroneous-use.\\
10154     In-the~ \@@_full_name_env: ,~you-must-use-the-key~
10155     'last-col'~without-value.\\
10156     However,~you-can-go-on-for-this-time~
10157     (the-value~' \l_keys_value_tl '~will-be-ignored).
10158   }
10159   \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10160   {
10161     Erroneous-use. \\
10162     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10163     'last-col'~without-value. \\
10164     However,~you-can-go-on-for-this-time~
10165     (the-value~' \l_keys_value_tl '~will-be-ignored).
10166   }
10167   \@@_msg_new:nn { Block-too-large-1 }
10168   {
10169     Block-too-large. \\
10170     You-try-to-draw-a-block-in-the-cell~#1-#2-of-your-matrix-but-the-matrix-is~
10171     too-small-for-that-block. \\
10172     This-block-and-maybe-others-will-be-ignored.
10173   }
10174   \@@_msg_new:nn { Block-too-large-2 }
10175   {
10176     Block-too-large. \\
10177     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10178     \g_@@_static_num_of_col_int \
10179     columns~but-you-use-only~ \int_use:N \c@jCol \ and-that's-why-a-block~
10180     specified-in-the-cell~#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
10181     (&)~at-the-end-of-the-first-row-of-your~ \@@_full_name_env: . \\
10182     This-block-and-maybe-others-will-be-ignored.
10183   }
10184   \@@_msg_new:nn { unknown-column-type }
10185   {
10186     Bad-column-type. \\
10187     The-column-type~'#1'~in-your~ \@@_full_name_env: \
10188     is-unknown. \\
10189     This-error-is-fatal.
10190   }
10191   \@@_msg_new:nn { unknown-column-type-multicolumn }
10192   {
10193     Bad-column-type. \\
10194     The-column-type~'#1'~in-the-command~\token_to_str:N \multicolumn \
10195     ~of-your~ \@@_full_name_env: \
10196     is-unknown. \\
10197     This-error-is-fatal.
10198   }

```

```

10199 \@@_msg_new:nn { unknown-column-type-S }
10200 {
10201   Bad-column-type. \\\
10202   The-column-type-'S'~in-your- \@@_full_name_env: \ is-unknown. \\\
10203   If-you-want-to-use-the-column-type-'S'~of-siunitx,-you-should-
10204   load-that-package. \\\
10205   This-error-is-fatal.
10206 }
10207 \@@_msg_new:nn { unknown-column-type-S-multicolumn }
10208 {
10209   Bad-column-type. \\\
10210   The-column-type-'S'~in-the-command~\token_to_str:N \multicolumn \
10211   of~your~ \@@_full_name_env: \ is-unknown. \\\
10212   If-you-want-to-use-the-column-type-'S'~of-siunitx,-you-should-
10213   load-that-package. \\\
10214   This-error-is-fatal.
10215 }
10216 \@@_msg_new:nn { tabularnote-forbidden }
10217 {
10218   Forbidden-command. \\\
10219   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10220   ~here.~This-command-is-available-only-in~
10221   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10222   the-argument-of-a-command~\token_to_str:N \caption \ included-
10223   in-an-environment~\{table\}. \\\
10224   This-command-will-be-ignored.
10225 }
10226 \@@_msg_new:nn { borders-forbidden }
10227 {
10228   Forbidden-key.\\
10229   You-can't-use-the-key~'borders'~of-the-command~ \token_to_str:N \Block \
10230   because-the-option~'rounded-corners'~
10231   is~in~force~with~a~non-zero~value.\\
10232   This-key-will-be-ignored.
10233 }
10234 \@@_msg_new:nn { bottomrule-without-booktabs }
10235 {
10236   booktabs-not-loaded.\\
10237   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
10238   loaded~'booktabs'.\\
10239   This-key-will-be-ignored.
10240 }
10241 \@@_msg_new:nn { enumitem-not-loaded }
10242 {
10243   enumitem-not-loaded. \\\
10244   You-can't-use-the-command~ \token_to_str:N \tabularnote \
10245   ~because-you-haven't~loaded~'enumitem'. \\\
10246   All-the-commands~ \token_to_str:N \tabularnote \ will-be-
10247   ignored-in-the-document.
10248 }
10249 \@@_msg_new:nn { tikz-without-tikz }
10250 {
10251   Tikz-not-loaded. \\\
10252   You-can't-use-the-key~'tikz'~here~because-Tikz-is-not~
10253   loaded.~If-you-go-on,~that-key-will-be-ignored.
10254 }
10255 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
10256 {
10257   Tikz-not-loaded. \\\
10258   You-have-used-the-key~'tikz'~in-the-definition-of-a~
10259   customized-line~(with~'custom-line')~but~tikz-is-not-loaded.~

```

```

10260     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10261     use~that~custom~line.
10262 }

10263 \@@_msg_new:nn { tikz-in~borders~without~tikz }
10264 {
10265     Tikz~not~loaded. \\
10266     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10267     command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10268     That~key~will~be~ignored.
10269 }

10270 \@@_msg_new:nn { color-in~custom~line~with~tikz }
10271 {
10272     Erroneous~use. \\
10273     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10274     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10275     The~key~'color'~will~be~discarded.
10276 }

10277 \@@_msg_new:nn { Wrong~last~row }
10278 {
10279     Wrong~number. \\
10280     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10281     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10282     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10283     last~row~but~you~should~correct~your~code.~You~can~avoid~this~
10284     problem~by~using~'last~row'~without~value~(more~compilations~
10285     might~be~necessary).
10286 }

10287 \@@_msg_new:nn { Yet~in~env }
10288 {
10289     Nested~environments. \\
10290     Environments~of~nicematrix~can't~be~nested. \\
10291     This~error~is~fatal.
10292 }

10293 \@@_msg_new:nn { Outside~math~mode }
10294 {
10295     Outside~math~mode. \\
10296     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10297     (and~not~in~ \token_to_str:N \vcenter ). \\
10298     This~error~is~fatal.
10299 }

10300 \@@_msg_new:nn { One~letter~allowed }
10301 {
10302     Bad~name. \\
10303     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10304     you~have~used~' \l_keys_value_tl '. \\
10305     It~will~be~ignored.
10306 }

10307 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10308 {
10309     Environment~\{TabularNote\}~forbidden. \\
10310     You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10311     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10312     This~environment~\{TabularNote\}~will~be~ignored.
10313 }

10314 \@@_msg_new:nn { varwidth~not~loaded }
10315 {
10316     varwidth~not~loaded. \\
10317     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10318     loaded. \\
10319     Your~column~will~behave~like~'p'.

```

```

10320 }
10321 \@@_msg_new:nn { varwidth-not-loaded-in-X }
10322 {
10323   varwidth-not-loaded.\
10324   You-can't-use-the-key~'V'~in-your-column~'X'~
10325   because~'varwidth'~is-not-loaded.\
10326   It~will~be~ignored. \
10327 }
10328 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10329 {
10330   Unknown~key.\
10331   Your~key~' \l_keys_key_str '~is-unknown~for~a~rule.\
10332   \c_@@_available_keys_str
10333 }
10334 {
10335   The-available-keys-are~(in~alphabetic~order):~
10336   color,~
10337   dotted,~
10338   multiplicity,~
10339   sep-color,~
10340   tikz,~and~total-width.
10341 }
10342
10343 \@@_msg_new:nnn { Unknown~key~for~Block }
10344 {
10345   Unknown~key. \
10346   The~key~' \l_keys_key_str '~is-unknown~for~the~command~
10347   \token_to_str:N \Block . \
10348   It~will~be~ignored. \
10349   \c_@@_available_keys_str
10350 }
10351 {
10352   The-available-keys-are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10353   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10354   opacity,~rounded-corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10355   and~vlines.
10356 }
10357 \@@_msg_new:nnn { Unknown~key~for~Brace }
10358 {
10359   Unknown~key.\
10360   The~key~' \l_keys_key_str '~is-unknown~for~the~commands~
10361   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \
10362   It~will~be~ignored. \
10363   \c_@@_available_keys_str
10364 }
10365 {
10366   The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
10367   right-shorten,~shorten~(which~fixes~both~left~shorten~and~
10368   right-shorten)~and~yshift.
10369 }
10370 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10371 {
10372   Unknown~key.\
10373   The~key~' \l_keys_key_str '~is-unknown.\
10374   It~will~be~ignored. \
10375   \c_@@_available_keys_str
10376 }
10377 {
10378   The-available-keys-are~(in~alphabetic~order):~
10379   delimiters/color,~
10380   rules~(with~the~subkeys~'color'~and~'width'),~
10381   sub-matrix~(several~subkeys)~

```



```

10382     and~xdots~(several~subkeys).~
10383     The~latter~is~for~the~command~ \token_to_str:N \line .
10384 }

10385 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10386 {
10387     Unknown~key.\\
10388     The~key~' \l_keys_key_str '~is~unknown.\\
10389     It~will~be~ignored. \\
10390     \c_@@_available_keys_str
10391 }
10392 {
10393     The~available~keys~are~(in~alphabetic~order):~
10394     create~cell~nodes,~
10395     delimiters/color~and~
10396     sub~matrix~(several~subkeys).
10397 }

10398 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10399 {
10400     Unknown~key.\\
10401     The~key~' \l_keys_key_str '~is~unknown.\\
10402     That~key~will~be~ignored. \\
10403     \c_@@_available_keys_str
10404 }
10405 {
10406     The~available~keys~are~(in~alphabetic~order):~
10407     'delimiters/color',~
10408     'extra~height',~
10409     'hlines',~
10410     'hvlines',~
10411     'left~xshift',~
10412     'name',~
10413     'right~xshift',~
10414     'rules'~(with~the~subkeys~'color'~and~'width'),~
10415     'slim',~
10416     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10417     and~'right~xshift').\\
10418 }

10419 \@@_msg_new:nnn { Unknown~key~for~notes }
10420 {
10421     Unknown~key.\\
10422     The~key~' \l_keys_key_str '~is~unknown.\\
10423     That~key~will~be~ignored. \\
10424     \c_@@_available_keys_str
10425 }
10426 {
10427     The~available~keys~are~(in~alphabetic~order):~
10428     bottomrule,~
10429     code~after,~
10430     code~before,~
10431     detect~duplicates,~
10432     enumitem~keys,~
10433     enumitem~keys~para,~
10434     para,~
10435     label~in~list,~
10436     label~in~tabular~and~
10437     style.
10438 }

10439 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10440 {
10441     Unknown~key.\\
10442     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10443     \token_to_str:N \RowStyle . \\

```

```

10444     That~key~will~be~ignored. \\
10445     \c_@@_available_keys_str
10446 }
10447 {
10448     The~available~keys~are~(in~alphabetic~order):~
10449     bold,~
10450     cell-space-top-limit,~
10451     cell-space-bottom-limit,~
10452     cell-space-limits,~
10453     color,~
10454     fill~(alias:~rowcolor),~
10455     nb-rows,~
10456     opacity~and~
10457     rounded-corners.
10458 }
10459 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10460 {
10461     Unknown~key.\\
10462     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10463     \token_to_str:N \NiceMatrixOptions . \\
10464     That~key~will~be~ignored. \\
10465     \c_@@_available_keys_str
10466 }
10467 {
10468     The~available~keys~are~(in~alphabetic~order):~
10469     &-in-blocks,~
10470     allow-duplicate-names,~
10471     ampersand-in-blocks,~
10472     caption-above,~
10473     cell-space-bottom-limit,~
10474     cell-space-limits,~
10475     cell-space-top-limit,~
10476     code-for-first-col,~
10477     code-for-first-row,~
10478     code-for-last-col,~
10479     code-for-last-row,~
10480     corners,~
10481     custom-key,~
10482     create-extra-nodes,~
10483     create-medium-nodes,~
10484     create-large-nodes,~
10485     custom-line,~
10486     delimiters~(several~subkeys),~
10487     end-of-row,~
10488     first-col,~
10489     first-row,~
10490     hlines,~
10491     hvlines,~
10492     hvlines-except-borders,~
10493     last-col,~
10494     last-row,~
10495     left-margin,~
10496     light-syntax,~
10497     light-syntax-expanded,~
10498     matrix/columns-type,~
10499     no-cell-nodes,~
10500     notes~(several~subkeys),~
10501     nullify-dots,~
10502     pgf-node-code,~
10503     renew-dots,~
10504     renew-matrix,~
10505     respect-arraystretch,~
10506     rounded-corners,~

```

```

10507     right-margin,~
10508     rules~(with~the~subkeys~'color'~and~'width'),~
10509     small,~
10510     sub-matrix~(several~subkeys),~
10511     vl_lines,~
10512     xdots~(several~subkeys).
10513 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10514 \@@_msg_new:nnn { Unknown-key-for~NiceArray }
10515 {
10516     Unknown-key.\\
10517     The~key~' \l_keys_key_str '~is-unknown-for~the~environment~
10518     \{NiceArray\}. \\
10519     That~key~will~be~ignored. \\
10520     \c_@@_available_keys_str
10521 }
10522 {
10523     The~available~keys~are~(in~alphabetic~order):~
10524     &in-blocks,~
10525     ampersand-in-blocks,~
10526     b,~
10527     baseline,~
10528     c,~
10529     cell-space-bottom-limit,~
10530     cell-space-limits,~
10531     cell-space-top-limit,~
10532     code-after,~
10533     code-for-first-col,~
10534     code-for-first-row,~
10535     code-for-last-col,~
10536     code-for-last-row,~
10537     columns-width,~
10538     corners,~
10539     create-extra-nodes,~
10540     create-medium-nodes,~
10541     create-large-nodes,~
10542     extra-left-margin,~
10543     extra-right-margin,~
10544     first-col,~
10545     first-row,~
10546     hlines,~
10547     hvlines,~
10548     hvlines-except-borders,~
10549     last-col,~
10550     last-row,~
10551     left-margin,~
10552     light-syntax,~
10553     light-syntax-expanded,~
10554     name,~
10555     no-cell-nodes,~
10556     nullify-dots,~
10557     pgf-node-code,~
10558     renew-dots,~
10559     respect-arraystretch,~
10560     right-margin,~
10561     rounded-corners,~
10562     rules~(with~the~subkeys~'color'~and~'width'),~
10563     small,~
10564     t,~
10565     vl_lines,~
10566     xdots/color,~
10567     xdots/shorten-start,~

```

```

10568     xdots/shorten-end,~
10569     xdots/shorten-and~
10570     xdots/line-style.
10571 }

```

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`).

```

10572 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10573 {
10574   Unknown~key.\\
10575   The~key~' \l_keys_key_str '~is~unknown~for~the~
10576   \@@_full_name_env: . \\
10577   That~key~will~be~ignored. \\
10578   \c_@@_available_keys_str
10579 }
10580 {
10581   The~available~keys~are~(in~alphabetic~order):~
10582   &-in-blocks,~
10583   ampersand-in-blocks,~
10584   b,~
10585   baseline,~
10586   c,~
10587   cell-space-bottom-limit,~
10588   cell-space-limits,~
10589   cell-space-top-limit,~
10590   code-after,~
10591   code-for-first-col,~
10592   code-for-first-row,~
10593   code-for-last-col,~
10594   code-for-last-row,~
10595   columns-type,~
10596   columns-width,~
10597   corners,~
10598   create-extra-nodes,~
10599   create-medium-nodes,~
10600   create-large-nodes,~
10601   extra-left-margin,~
10602   extra-right-margin,~
10603   first-col,~
10604   first-row,~
10605   hlines,~
10606   hvlines,~
10607   hvlines-except-borders,~
10608   l,~
10609   last-col,~
10610   last-row,~
10611   left-margin,~
10612   light-syntax,~
10613   light-syntax-expanded,~
10614   name,~
10615   no-cell-nodes,~
10616   nullify-dots,~
10617   pgf-node-code,~
10618   r,~
10619   renew-dots,~
10620   respect-arraystretch,~
10621   right-margin,~
10622   rounded-corners,~
10623   rules~(with~the~subkeys~'color'~and~'width'),~
10624   small,~
10625   t,~
10626   vlides,~
10627   xdots/color,~
10628   xdots/shorten-start,~

```

```

10629     xdots/shorten-end,~
10630     xdots/shorten-and~
10631     xdots/line-style.
10632 }
10633 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10634 {
10635     Unknown~key.\\
10636     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10637     \{NiceTabular\}. \\
10638     That~key~will~be~ignored. \\
10639     \c_@@_available_keys_str
10640 }
10641 {
10642     The~available~keys~are~(in~alphabetic~order):~
10643     &~in~blocks,~
10644     ampersand~in~blocks,~
10645     b,~
10646     baseline,~
10647     c,~
10648     caption,~
10649     cell-space-bottom-limit,~
10650     cell-space-limits,~
10651     cell-space-top-limit,~
10652     code-after,~
10653     code-for-first-col,~
10654     code-for-first-row,~
10655     code-for-last-col,~
10656     code-for-last-row,~
10657     columns-width,~
10658     corners,~
10659     custom-line,~
10660     create-extra-nodes,~
10661     create-medium-nodes,~
10662     create-large-nodes,~
10663     extra-left-margin,~
10664     extra-right-margin,~
10665     first-col,~
10666     first-row,~
10667     hlines,~
10668     hvlines,~
10669     hvlines-except-borders,~
10670     label,~
10671     last-col,~
10672     last-row,~
10673     left-margin,~
10674     light-syntax,~
10675     light-syntax-expanded,~
10676     name,~
10677     no-cell-nodes,~
10678     notes~(several~subkeys),~
10679     nullify-dots,~
10680     pgf-node-code,~
10681     renew-dots,~
10682     respect-arraystretch,~
10683     right-margin,~
10684     rounded-corners,~
10685     rules~(with~the~subkeys~'color'~and~'width'),~
10686     short-caption,~
10687     t,~
10688     tabularnote,~
10689     vlides,~
10690     xdots/color,~
10691     xdots/shorten-start,~

```

```

10692     xdots/shorten-end,~
10693     xdots/shorten-and~
10694     xdots/line-style.
10695 }

10696 \@@_msg_new:nnn { Duplicate-name }
10697 {
10698     Duplicate-name.\\
10699     The-name~' \l_keys_value_tl '~is-already-used-and-you-shouldn't-use~
10700     the-same-environment-name-twice.~You-can-go-on,~but,~
10701     maybe,~you-will-have-incorrect-results-especially~
10702     if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10703     message-again,~use-the-key~'allow-duplicate-names'~in~
10704     ' \token_to_str:N \NiceMatrixOptions '~\\
10705     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10706     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10707 }
10708 {
10709     The-names-already-defined-in-this-document-are:~
10710     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10711 }

10712 \@@_msg_new:nn { Option-auto-for-columns-width }
10713 {
10714     Erroneous-use.\\
10715     You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10716     That-key-will-be-ignored.
10717 }

10718 \@@_msg_new:nn { NiceTabularX-without-X }
10719 {
10720     NiceTabularX-without-X.\\
10721     You-should-not-use~\{NiceTabularX\}~without-X-columns.\\
10722     However,~you-can-go-on.
10723 }

10724 \@@_msg_new:nn { Preamble-forgotten }
10725 {
10726     Preamble-forgotten.\\
10727     You-have-probably-forgotten-the-preamble-of-your~
10728     \@@_full_name_env: . \\
10729     This-error-is-fatal.
10730 }

10731 \@@_msg_new:nn { Invalid-col-number }
10732 {
10733     Invalid-column-number.\\
10734     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10735     specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.
10736 }

10737 \@@_msg_new:nn { Invalid-row-number }
10738 {
10739     Invalid-row-number.\\
10740     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10741     specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.
10742 }

10743 \@@_define_com:NNN p ( )
10744 \@@_define_com:NNN b [ ]
10745 \@@_define_com:NNN v | |
10746 \@@_define_com:NNN V \l \l
10747 \@@_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	25
7	The options	26
8	Important code used by <code>{NiceArrayWithDelims}</code>	37
9	The <code>\CodeBefore</code>	51
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	76
13	The environment <code>{NiceMatrix}</code> and its variants	94
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	95
15	After the construction of the array	96
16	We draw the dotted lines	103
17	The actual instructions for drawing the dotted lines with Tikz	118
18	User commands available in the new environments	123
19	The command <code>\line</code> accessible in code-after	129
20	The command <code>\RowStyle</code>	131
21	Colors of cells, rows and columns	134
22	The vertical and horizontal rules	146
23	The empty corners	162
24	The environment <code>{NiceMatrixBlock}</code>	164
25	The extra nodes	166
26	The blocks	170
27	How to draw the dotted lines transparently	195
28	Automatic arrays	195
29	The redefinition of the command <code>\dotfill</code>	197
30	The command <code>\diagbox</code>	197

31	The keyword <code>\CodeAfter</code>	198
32	The delimiters in the preamble	199
33	The command <code>\SubMatrix</code>	200
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	209
35	The commands <code>HBrace</code> et <code>VBrace</code>	212
36	The command <code>TikzEveryCell</code>	215
37	The command <code>\ShowCellNames</code>	216
38	We process the options at package loading	218
39	About the package underscore	220
40	Error messages of the package	220