

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

January 12, 2026

## Abstract

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

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

---

\*This document corresponds to the version 7.5a of `nicematrix`, at the date of 2026/01/05.

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

```

20 \RequirePackage { amsmath }

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

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

```

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

```

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

81 \tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
82 \tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }

```

The following token list will be used for definitions of user commands (with `\NewDocumentCommand`) with an embellishment using an *underscore* (there may be problems because of the catcode of the underscore).

```

83 \tl_new:N \l_@@_argspec_tl

```

```

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

92 \hook_gput_code:nnn { begindocument } { . }
93 {
94   \IfPackageLoadedTF { tikz }
95   {

```

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.

```

96   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
97   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
98 }
99 {
100   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
101   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
102 }
103 }

```

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`).

```

104 \IfClassLoadedTF { revtex4-1 }
105 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
106 {
107   \IfClassLoadedTF { revtex4-2 }
108   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
109   {

```

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

```

110   \cs_if_exist:NT \rvtx@ifformat@geq
111   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112   { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
113 }
114 }

```

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.

```

115 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
116 {
117   \iow_now:Nn \@mainaux
118   {
119     \ExplSyntaxOn
120     \cs_if_free:NT \pgfsyspdfmark
121     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
122     \ExplSyntaxOff
123   }
124   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
125 }

```

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.

```

126 \ProvideDocumentCommand \iddots { }
127 {
128   \mathinner
129   {
130     \mkern 1 mu
131     \box_move_up:nn { 1 pt } { \hbox { . } }
132     \mkern 2 mu
133     \box_move_up:nn { 4 pt } { \hbox { . } }
134     \mkern 2 mu
135     \box_move_up:nn { 7 pt }
136     { \vbox:n { \kern 7 pt \hbox { . } } }
137     \mkern 1 mu
138   }
139 }

```

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.

```

140 \hook_gput_code:nnn { begindocument } { . }
141 {
142   \IfPackageLoadedT { booktabs }
143   { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
144 }
145 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
146 {
147   \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`).

```

148   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
149   {

```

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

```

150     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
151     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
152   }
153 }

```

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

```

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

```

```

171     }
172 }

```

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).

```

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

```

178     }
179 }
180 {
181     \cs_new_protected:Npn \@@_revert_colortbl: { }
182     \cs_new_protected:Npn \@@_replace_columncolor:
183     { \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.

```

184 \def \CT@arc@ { }
185 \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
186 \def \CT@arc@ #1 #2
187 {
188     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
189     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
190 }

```

Idem for `\CT@drs@`.

```

191 \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
192 \def \CT@drs@ #1 #2
193 {
194     \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
195     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
196 }
197 \def \hline
198 {
199     \noalign { \ifnum 0 = ` } \fi
200     \cs_set_eq:NN \hskip \vskip
201     \cs_set_eq:NN \vrule \hrule
202     \cs_set_eq:NN \@width \@height
203     { \CT@arc@ \vline }
204     \futurelet \reserved@a
205     \@xhline
206 }
207 }
208 }

```

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.

```

209 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
210 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
211 {
212     \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
213     \int_compare:nNnT { #1 } > { \c_one_int }
214     { \multispan { \int_eval:n { #1 - 1 } } & }
215     \multispan { \int_eval:n { #2 - #1 + 1 } }
216     {
217         \CT@arc@
218         \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>

```
219     \skip_horizontal:N \c_zero_dim
220 }
```

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`.

```
221     \everycr { }
222     \cr
223     \noalign { \skip_vertical:n { - \arrayrulewidth } }
224 }
```

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.

```
225 \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`.

```
226 { \@@_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*.

```
227 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
228 \cs_generate_variant:Nn \@@_cline_i:nn { e }
229 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
230 {
231     \tl_if_empty:nTF { #3 }
232     { \@@_cline_iii:w #1|#2-#2 \q_stop }
233     { \@@_cline_ii:w #1|#2-#3 \q_stop }
234 }
235 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
236 { \@@_cline_iii:w #1|#2-#3 \q_stop }
237 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
238 {
```

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).

```
239     \int_compare:nNnT { #1 } < { #2 }
240     { \multispan { \int_eval:n { #2 - #1 } } & }
241     \multispan { \int_eval:n { #3 - #2 + 1 } }
242     {
243         \CT@arc@
244         \leaders \hrule \@height \arrayrulewidth \hfill
245         \skip_horizontal:N \c_zero_dim
246     }
```

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

```
247     \peek_meaning_remove_ignore_spaces:NNTF \cline
248     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
249     { \everycr { } \cr }
250 }
```

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

```
251 \cs_set:Nn \@@_math_toggle: { $ } % $
```

---

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

```

252 \cs_new_protected:Npn \@@_set_CTarc:n #1
253 {
254   \tl_if_blank:nF { #1 }
255   {
256     \tl_if_head_eq_meaning:nNTF { #1 } [
257       { \def \CT@arc@ { \color #1 } }
258       { \def \CT@arc@ { \color { #1 } } }
259     ]
260   }
261 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

262 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
263 {
264   \tl_if_head_eq_meaning:nNTF { #1 } [
265     { \def \CT@drsc@ { \color #1 } }
266     { \def \CT@drsc@ { \color { #1 } } }
267   ]

```

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

```

268 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
269 {
270   \tl_if_head_eq_meaning:nNTF { #2 } [
271     { #1 #2 }
272     { #1 { #2 } }
273   ]
274 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

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

```

```

278 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
279 {
280   \tl_set_rescan:Nno
281     #1
282     {
283       \char_set_catcode_other:N >
284       \char_set_catcode_other:N <
285     }
286   #1
287 }

```

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

```

288 \dim_new:N \l_@@_tmpc_dim
289 \dim_new:N \l_@@_tmpd_dim
290 \tl_new:N \l_@@_tmpc_tl
291 \tl_new:N \l_@@_tmpd_tl
292 \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.

```
293 \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).

```
294 \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.

```
295 \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.

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

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

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

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

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

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

```
300 \bool_new:N \g_@@_delims_bool
301 \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.

```
302 \bool_new:N \l_@@_preamble_bool
303 \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.

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

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

```
305 \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.

```
306 \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).

```
307 \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.).

```
308 \dim_new:N \l_@@_col_width_dim
309 \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.

```
310 \int_new:N \g_@@_row_total_int
311 \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).

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

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

```
313 \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.

```
314 \tl_new:N \l_@@_hpos_cell_tl
315 \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`.

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

Idem for the mono-row blocks.

```
317 \dim_new:N \g_@@_blocks_ht_dim
318 \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`).

```
319 \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`.

```
320 \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.

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

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

```
322 \bool_new:N \l_@@_notes_detect_duplicates_bool
323 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```

324 \bool_new:N \l_@@_initial_open_bool
325 \bool_new:N \l_@@_final_open_bool
326 \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.

```

327 \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).

```

328 \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).

```

329 \tl_new:N \l_@@_rule_color_tl

```

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

```

330 \bool_new:N \g_@@_rotate_bool

```

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

```

331 \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.

```

332 \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`).

```

333 \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`.

```

334 \bool_new:N \g_@@_V_of_X_bool

```

```

335 \bool_new:N \g_@@_caption_finished_bool

```

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

```

336 \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 }`).

```

337 \tl_new:N \g_@@_aux_tl

```

During the second run, if information concerning the current environment has been found in the `aux` file, the following flag will be raised. It will be used, for instance to disable several constructions (continuous dotted lines, and colored backgrounds) during the first compilation (in order to speed up it).

```

338 \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.

```

339 \seq_new:N \g_@@_size_seq

```

```

340 \tl_new:N \g_@@_left_delim_tl

```

```

341 \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}`).

```
342 \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`).

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

For `\multicolumn`.

```
344 \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`.

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

```
346 \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 `..`.

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

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

```
349 \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`).

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

```
351 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
352 {
```

```
353   \if_mode_math: \else:
```

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

```
355   \fi:
```

```
356 }
```

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.

```
357 \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”.

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

```
359 \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`).

```
360 \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*.

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

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

```
363 \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)`.

```

364 \cs_new:Npn \@@_full_name_env:
365 {
366   \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
367   { command \space \c_backslash_str \g_@@_name_env_str }
368   { environment \space \{ \g_@@_name_env_str \} }
369 }

```

```

370 \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.

```

371 \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.

```

372 \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.

```

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

```

375 \tl_new:N \g_@@_pre_code_after_tl
376 \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.

```

377 \bool_new:N \l_@@_in_code_after_bool

```

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

```

378 \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.

```

379 \int_new:N \l_@@_old_iRow_int
380 \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).

```

381 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

382 \tl_new:N \l_@@_rules_color_tl

```

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

```
383 \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.

```
384 \bool_new:N \l_@@_X_columns_aux_bool
385 \dim_new:N \l_@@_X_columns_dim
```

```
386 \dim_new:N \l_@@_brace_shift_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.

```
387 \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).

```
388 \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).

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

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

```
391 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

```
403 \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”.

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

```
405 \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.

```
406 \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!).

```
407 \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.

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

They will be added to `\g_@@_pos_of_blocks_seq` after the computation of the “empty corners”.

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by `\Cdots`, `\Vdots`, `\Ddots`, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: `{imin}{jmin}{imax}{jmax}{ name}`.

```
409 \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.

```
410 \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`).

```
411 \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`).

```
412 \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.

```
413 \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).

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

```
415 \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.

```
416 \int_new:N \g_@@_ddots_int
417 \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.

```
418 \dim_new:N \g_@@_delta_x_one_dim
419 \dim_new:N \g_@@_delta_y_one_dim
420 \dim_new:N \g_@@_delta_x_two_dim
421 \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).

```
422 \int_new:N \l_@@_row_min_int
423 \int_new:N \l_@@_row_max_int
424 \int_new:N \l_@@_col_min_int
425 \int_new:N \l_@@_col_max_int

426 \int_new:N \l_@@_initial_i_int
427 \int_new:N \l_@@_initial_j_int
428 \int_new:N \l_@@_final_i_int
429 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
430 \int_new:N \l_@@_start_int
431 \int_set_eq:NN \l_@@_start_int \c_one_int
432 \int_new:N \l_@@_end_int
433 \int_new:N \l_@@_local_start_int
434 \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.

```
435 \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).

```
436 \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`.

```
437 \tl_new:N \l_@@_fill_tl
438 \tl_new:N \l_@@_opacity_tl
439 \tl_new:N \l_@@_draw_tl
440 \seq_new:N \l_@@_tikz_seq
441 \clist_new:N \l_@@_borders_clist
442 \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.

```
443 \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`.

```
444 \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.

```
445 \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.

```
446 \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`).

```
447 \str_new:N \l_@@_hpos_block_str
448 \str_set:Nn \l_@@_hpos_block_str { c }
449 \bool_new:N \l_@@_hpos_of_block_cap_bool
450 \bool_new:N \l_@@_p_block_bool
```

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

```
451 \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).

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

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

```
453 \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).

```
454 \bool_new:N \l_@@_vlines_block_bool
455 \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.

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

457 \dim_new:N \l_@@_submatrix_extra_height_dim
458 \dim_new:N \l_@@_submatrix_left_xshift_dim
459 \dim_new:N \l_@@_submatrix_right_xshift_dim
460 \clist_new:N \l_@@_hlines_clist
461 \clist_new:N \l_@@_vlines_clist
462 \clist_new:N \l_@@_submatrix_hlines_clist
463 \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}`).

```
464 \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.

```
465 \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`).

```
466 \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.

```
467 \int_new:N \l_@@_first_row_int
468 \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.

```
469 \int_new:N \l_@@_first_col_int
470 \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).

```
471 \int_new:N \l_@@_last_row_int
472 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
474 \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  $0$ .

```
475 \int_new:N \l_@@_last_col_int
476 \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.

```
477 \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`).

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

## Some utilities

```
479 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
480 {
```

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

```
481 \def \l_tmpa_tl { #1 }
482 \def \l_tmpb_tl { #2 }
483 }
```

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`.

```
484 \cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
485 {
486   \clist_if_in:NnF #1 { all }
487   {
488     \clist_clear:N \l_tmpa_clist
489     \clist_map_inline:Nn #1
490     {
491       \tl_if_head_eq_meaning:nNTF { ##1 } -
492       {
```

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.

```
493       \int_if_zero:nF { #2 }
494       {
495         \clist_put_right:Ne \l_tmpa_clist
496         { \int_eval:n { #2 + (##1) + 1 } }
497       }
498     }
499   }
```

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

```

500         \tl_if_in:nnTF { ##1 } { - }
501         { \@@_cut_on_hyphen:w ##1 \q_stop }
502         {

```

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

```

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

```

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.

```

513 \hook_gput_code:nnn { begindocument } { . }
514 {
515     \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
516     \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
517 }

```

## 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.

```
518 \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`.

```
519 \int_new:N \g_@@_tabularnote_int
520 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
521 \seq_new:N \g_@@_notes_seq
522 \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.

```
523 \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.

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

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

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

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

```
532 \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`.

```
533 \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`.

```
534 \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`).

```
535 \hook_gput_code:nnn { begindocument } { . }
536 {
537   \IfPackageLoadedTF { enumitem }
538   {
```

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.

```

539     \newlist { tabularnotes } { enumerate } { 1 }
540     \setlist [ tabularnotes ]
541     {
542         topsep = \c_zero_dim ,
543         noitemsep ,
544         leftmargin = * ,
545         align = left ,
546         labelsep = \c_zero_dim ,
547         label =
548             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
549     }
550     \newlist { tabularnotes* } { enumerate* } { 1 }
551     \setlist [ tabularnotes* ]
552     {
553         afterlabel = \nobreak ,
554         itemjoin = \quad ,
555         label =
556             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
557     }

```

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`.

```

558     \NewDocumentCommand \tabularnote { o m }
559     {
560         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } { \l_@@_in_env_bool }
561         {
562             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
563             { \@@_error:n { tabularnote~forbidden } }
564             {
565                 \bool_if:NTF \l_@@_in_caption_bool
566                 \@@_tabularnote_caption:nn
567                 \@@_tabularnote:nn
568                 { #1 } { #2 }
569             }
570         }
571     }
572 }
573 {
574     \NewDocumentCommand \tabularnote { o m }
575     { \@@_err_enumitem_not_loaded: }
576 }
577 }

578 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
579 {
580     \@@_error_or_warning:n { enumitem~not~loaded }
581     \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
582 }

583 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
584 { \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`.

```

585 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
586 {

```

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).

```

587     \int_zero:N \l_tmpa_int
588     \bool_if:NT \l_@@_notes_detect_duplicates_bool
589     {

```

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`.

```

590     \int_zero:N \l_tmpb_int
591     \seq_map_indexed_inline:Nn \g_@@_notes_seq
592     {
593         \@@_test_first_novaluennn ##2 { \int_incr:N \l_tmpb_int }
594         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
595         {
596             \tl_if_novaluenTF { #1 }
597             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
598             { \int_set:Nn \l_tmpa_int { ##1 } }
599             \seq_map_break:
600         }
601     }
602     \int_if_zero:nF { \l_tmpa_int }
603     { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
604 }
605 \int_if_zero:nT { \l_tmpa_int }
606 {
607     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
608     \tl_if_novaluenTF { #1 } { \int_gincr:N \c@tabularnote }
609 }
610 \seq_put_right:Ne \l_@@_notes_labels_seq
611 {
612     \tl_if_novaluenTF { #1 }
613     {
614         \@@_notes_format:n
615         {
616             \int_eval:n
617             {
618                 \int_if_zero:nTF { \l_tmpa_int }
619                 { \c@tabularnote }
620                 { \l_tmpa_int }
621             }
622         }
623     }
624     { #1 }
625 }
626 \peek_meaning:NF \tabularnote
627 {

```

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`.

```

628     \hbox_set:Nn \l_tmpa_box
629     {

```

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

```

630         \@@_notes_label_in_tabular:n
631         { \seq_use:Nn \l_@@_notes_labels_seq { , } }
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 { \seq_use:Nn \l_@@_notes_labels_seq { , } }
678         \seq_clear:N \l_@@_notes_labels_seq
679     }
680 }

```

```

681 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
682 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

## 6 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

**#1** is the name of the node which will be created; **#2** and **#3** are the coordinates of one of the corner of the rectangle; **#4** and **#5** are the coordinates of the opposite corner.

```

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

```

The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgf_rect_node:nnnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

707 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
708 {
709     \begin { pgfscope }
710     \pgfset
711     {
712         inner~sep = \c_zero_dim ,

```

```

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

```

## 7 The options

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

```

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

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

733 \bool_new:N \l_@@_caption_above_bool

```

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

734 \bool_new:N \l_@@_standard_cline_bool

```

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

```

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

```

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

```

737 \bool_new:N \l_@@_xdots_h_labels_bool

```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```

738 \dim_new:N \l_@@_xdots_inter_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the distance between a node (in fact an anchor of that node) and a dotted line (for real dotted lines, the actual distance may, of course, be a bit larger, depending of the exact position of the dots).

```

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

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```

748 \dim_new:N \l_@@_xdots_radius_dim
749 \hook_gput_code:nnn { begindocument } { . }
750 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

The unit is `em` and that's why we fix the dimension after the preamble.

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

```

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

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

```

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

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain **an integer** (which represents the number of the row to which align the array).

```

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

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

```

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

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

```

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

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

```

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

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

```

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

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

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

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

```
764 \cs_new_protected:Npn \@@_reset_arraystretch: { \def \arraystretch { 1 } }
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

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

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

```
766 \bool_new:N \g_@@_create_cell_nodes_bool
```

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

```
767 \str_new:N \l_@@_name_str
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is 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.

```

779 \bool_new:N \l_@@_delimiters_max_width_bool

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

```

```

833   color .value_required:n = true ,
834   radius .code:n =
835     \hook_gput_code:nnn { begindocument } { . }
836     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
837   radius .value_required:n = true ,
838   inter .code:n =
839     \hook_gput_code:nnn { begindocument } { . }
840     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
841   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}`.

```

842   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
843   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
844   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).

```

845   draw-first .code:n = \prg_do_nothing: ,
846   unknown .code:n =
847     \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
848 }

```

```

849 \keys_define:nn { nicematrix / rules }
850 {
851   color .tl_set:N = \l_@@_rules_color_tl ,
852   color .value_required:n = true ,
853   width .dim_set:N = \arrayrulewidth ,
854   width .value_required:n = true ,
855   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
856 }

```

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

```

857 \keys_define:nn { nicematrix / Global }
858 {
859   caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
860   show-cell-names .code:n = \@@_error_or_warning:n { show-cell-names } ,
861   color-inside .code:n = \@@_fatal:n { key~color~inside } ,
862   colortbl-like .code:n = \@@_fatal:n { key~color~inside } ,
863   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
864   ampersand-in-blocks .default:n = true ,
865   &-in-blocks .meta:n = ampersand-in-blocks ,
866   no-cell-nodes .code:n =
867     \bool_set_true:N \l_@@_no_cell_nodes_bool
868     \cs_set_protected:Npn \@@_node_cell:
869       { \set@color \box_use_drop:N \l_@@_cell_box } ,
870   no-cell-nodes .value_forbidden:n = true ,
871   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
872   rounded-corners .default:n = 4 pt ,
873   custom-line .code:n = \@@_custom_line:n { #1 } ,
874   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
875   rules .value_required:n = true ,
876   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
877   standard-cline .default:n = true ,
878   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
879   cell-space-top-limit .value_required:n = true ,
880   cell-space-top-limit+ .code:n =
881     \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
882   cell-space-top-limit+ .value_required:n = true ,

```

```

883 cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
884 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
885 cell-space-bottom-limit .value_required:n = true ,
886 cell-space-bottom-limit+ .code:n =
887   \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
888 cell-space-bottom-limit+ .value_required:n = true ,
889 cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
890 cell-space-limits .meta:n =
891   {
892     cell-space-top-limit = #1 ,
893     cell-space-bottom-limit = #1 ,
894   } ,
895 cell-space-limits .value_required:n = true ,
896 cell-space-limits+ .meta:n =
897   {
898     cell-space-top-limit += #1 ,
899     cell-space-bottom-limit += #1 ,
900   } ,
901 cell-space-limits+ .value_required:n = true ,
902 cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
903 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
904 light-syntax .code:n =
905   \bool_set_true:N \l_@@_light_syntax_bool
906   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
907 light-syntax .value_forbidden:n = true ,
908 light-syntax-expanded .code:n =
909   \bool_set_true:N \l_@@_light_syntax_bool
910   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
911 light-syntax-expanded .value_forbidden:n = true ,
912 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
913 end-of-row .value_required:n = true ,
914 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
915 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
916 last-row .int_set:N = \l_@@_last_row_int ,
917 last-row .default:n = -1 ,
918 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
919 code-for-first-col .value_required:n = true ,
920 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
921 code-for-last-col .value_required:n = true ,
922 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
923 code-for-first-row .value_required:n = true ,
924 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
925 code-for-last-row .value_required:n = true ,
926 hlines .clist_set:N = \l_@@_hlines_clist ,
927 vlines .clist_set:N = \l_@@_vlines_clist ,
928 hlines .default:n = all ,
929 vlines .default:n = all ,
930 vlines-in-sub-matrix .code:n =
931   {
932     \tl_if_single_token:nTF { #1 }
933     {
934       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
935       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

936     { \cs_set_eq:cN { @@_#1 : } \@@_make_preamble_vlism:n }
937   }
938   { \@@_error:n { One~letter~allowed } }
939 } ,
940 vlines-in-sub-matrix .value_required:n = true ,
941 hvlines .code:n =
942   {
943     \bool_set_true:N \l_@@_hvlines_bool
944     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl

```

```

945     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
946   } ,
947   hvlines .value_forbidden:n = true ,
948   hvlines-except-borders .code:n =
949   {
950     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
951     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
952     \bool_set_true:N \l_@@_hvlines_bool
953     \bool_set_true:N \l_@@_except_borders_bool
954   } ,
955   hvlines-except-borders .value_forbidden:n = true ,
956   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.

```

957   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
958   renew-dots .value_forbidden:n = true ,
959   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
960   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
961   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
962   create-extra-nodes .meta:n =
963   { create-medium-nodes , create-large-nodes } ,
964   left-margin .dim_set:N = \l_@@_left_margin_dim ,
965   left-margin .default:n = \arraycolsep ,
966   right-margin .dim_set:N = \l_@@_right_margin_dim ,
967   right-margin .default:n = \arraycolsep ,
968   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
969   margin .default:n = \arraycolsep ,
970   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
971   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
972   extra-margin .meta:n =
973   { extra-left-margin = #1 , extra-right-margin = #1 } ,
974   extra-margin .value_required:n = true ,
975   respect-arraystretch .code:n =
976   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
977   respect-arraystretch .value_forbidden:n = true ,
978   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
979   pgf-node-code .value_required:n = true
980 }

```

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

```

981 \keys_define:nn { nicematrix / environments }
982 {
983   corners .clist_set:N = \l_@@_corners_clist ,
984   corners .default:n = { NW , SW , NE , SE } ,
985   code-before .code:n =
986   {
987     \tl_if_empty:nF { #1 }
988     {
989       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
990       \bool_set_true:N \l_@@_code_before_bool
991     }
992   } ,
993   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}`.

```

994   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
995   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
996   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,

```



```

997     baseline .tl_set:N = \l_@@_baseline_tl ,
998     baseline .value_required:n = true ,
999     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)`.

```

1000     \str_if_eq:eeTF { #1 } { auto }
1001     { \bool_set_true:N \l_@@_auto_columns_width_bool }
1002     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
1003     columns-width .value_required:n = true ,
1004     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.

```

1005     \legacy_if:nF { measuring@ }
1006     {
1007         \str_set:Nc \l_@@_name_str { #1 }
1008         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
1009         { \@@_err_duplicate_names:n { #1 } }
1010         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
1011     } ,
1012     name .value_required:n = true ,
1013     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
1014     code-after .value_required:n = true ,
1015 }

1016 \cs_set:Npn \@@_err_duplicate_names:n #1
1017 { \@@_error:nn { Duplicate-name } { #1 } }

1018 \keys_define:nn { nicematrix / notes }
1019 {
1020     para .bool_set:N = \l_@@_notes_para_bool ,
1021     para .default:n = true ,
1022     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1023     code-before .value_required:n = true ,
1024     code-before+ .code:n =
1025         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1026     code-before+ .value_required:n = true ,
1027     code-before~+ .code:n =
1028         \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
1029     code-before~+ .value_required:n = true ,
1030     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1031     code-after .value_required:n = true ,
1032     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1033     bottomrule .default:n = true ,
1034     style .cs_set:Np = \@@_notes_style:n #1 ,
1035     style .value_required:n = true ,
1036     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1037     label-in-tabular .value_required:n = true ,
1038     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1039     label-in-list .value_required:n = true ,
1040     enumitem-keys .code:n =
1041     {
1042         \hook_gput_code:nnn { begindocument } { . }
1043         {
1044             \IfPackageLoadedT { enumitem }
1045             { \setlist* [ tabularnotes ] { #1 } }
1046         }
1047     } ,
1048     enumitem-keys .value_required:n = true ,
1049     enumitem-keys-para .code:n =
1050     {
1051         \hook_gput_code:nnn { begindocument } { . }
1052         {
1053             \IfPackageLoadedT { enumitem }

```

```

1054         { \setlist* [ tabularnotes* ] { #1 } }
1055     }
1056 },
1057 enumitem-keys-para .value_required:n = true ,
1058 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1059 detect-duplicates .default:n = true ,
1060 unknown .code:n =
1061     \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
1062 }
1063 \keys_define:nn { nicematrix / delimiters }
1064 {
1065     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1066     max-width .default:n = true ,
1067     color .tl_set:N = \l_@@_delimiters_color_tl ,
1068     color .value_required:n = true ,
1069 }

```

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

```

1070 \keys_define:nn { nicematrix }
1071 {
1072     NiceMatrixOptions .inherit:n =
1073         { nicematrix / Global } ,
1074     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1075     NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1076     NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1077     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1078     SubMatrix / rules .inherit:n = nicematrix / rules ,
1079     CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1080     CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1081     CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1082     NiceMatrix .inherit:n =
1083         {
1084             nicematrix / Global ,
1085             nicematrix / environments ,
1086         } ,
1087     NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1088     NiceMatrix / rules .inherit:n = nicematrix / rules ,
1089     NiceTabular .inherit:n =
1090         {
1091             nicematrix / Global ,
1092             nicematrix / environments
1093         } ,
1094     NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1095     NiceTabular / rules .inherit:n = nicematrix / rules ,
1096     NiceTabular / notes .inherit:n = nicematrix / notes ,
1097     NiceArray .inherit:n =
1098         {
1099             nicematrix / Global ,
1100             nicematrix / environments ,
1101         } ,
1102     NiceArray / xdots .inherit:n = nicematrix / xdots ,
1103     NiceArray / rules .inherit:n = nicematrix / rules ,
1104     pNiceArray .inherit:n =
1105         {
1106             nicematrix / Global ,
1107             nicematrix / environments ,
1108         } ,
1109     pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1110     pNiceArray / rules .inherit:n = nicematrix / rules ,
1111 }

```

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

```

1112 \keys_define:nn { nicematrix / NiceMatrixOptions }
1113 {
1114   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1115   delimiters / color .value_required:n = true ,
1116   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1117   delimiters / max-width .default:n = true ,
1118   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1119   delimiters .value_required:n = true ,
1120   width .dim_set:N = \l_@@_width_dim ,
1121   width .value_required:n = true ,
1122   last-col .code:n =
1123     \tl_if_empty:nF { #1 }
1124     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1125     \int_zero:N \l_@@_last_col_int ,
1126   small .bool_set:N = \l_@@_small_bool ,
1127   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.

```

1128   renew-matrix .code:n = \@@_renew_matrix: ,
1129   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}`.

```

1130   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.

```

1131   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)`.

```

1132     \str_if_eq:eeTF { #1 } { auto }
1133     { \@@_error:n { Option-auto-for-columns-width } }
1134     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1135   allow-duplicate-names .code:n =
1136     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1137   allow-duplicate-names .value_forbidden:n = true ,
1138   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1139   notes .value_required:n = true ,
1140   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1141   sub-matrix .value_required:n = true ,
1142   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1143   matrix / columns-type .value_required:n = true ,
1144   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1145   caption-above .default:n = true ,
1146   unknown .code:n =
1147     \@@_unknown_key:nn
1148     { nicematrix / Global , nicematrix / NiceMatrixOptions }
1149     { Unknown-key-for-NiceMatrixOptions }
1150 }

```

`\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.

```

1151 \NewDocumentCommand \NiceMatrixOptions { m }
1152 { \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.

```

1153 \keys_define:nn { nicematrix / NiceMatrix }
1154 {
1155   last-col .code:n = \tl_if_empty:nTF { #1 }
1156             {
1157               \bool_set_true:N \l_@@_last_col_without_value_bool
1158               \int_set:Nn \l_@@_last_col_int { -1 }
1159             }
1160             { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1161   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1162   columns-type .value_required:n = true ,
1163   l .meta:n = { columns-type = l } ,
1164   r .meta:n = { columns-type = r } ,
1165   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1166   delimiters / color .value_required:n = true ,
1167   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1168   delimiters / max-width .default:n = true ,
1169   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1170   delimiters .value_required:n = true ,
1171   small .bool_set:N = \l_@@_small_bool ,
1172   small .value_forbidden:n = true ,
1173   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1174 }

```

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

```

1175 \keys_define:nn { nicematrix / NiceArray }
1176 {

```

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.

```

1177   small .bool_set:N = \l_@@_small_bool ,
1178   small .value_forbidden:n = true ,
1179   last-col .code:n = \tl_if_empty:nF { #1 }
1180                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1181                 \int_zero:N \l_@@_last_col_int ,
1182   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1183   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1184   unknown .code:n =
1185     \@@_unknown_key:nn
1186     { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments }
1187     { Unknown~key~for~NiceArray }
1188 }
1189 \keys_define:nn { nicematrix / pNiceArray }
1190 {
1191   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1192   last-col .code:n = \tl_if_empty:nF { #1 }
1193                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1194                 \int_zero:N \l_@@_last_col_int ,
1195   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1196   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1197   delimiters / color .value_required:n = true ,
1198   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1199   delimiters / max-width .default:n = true ,
1200   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1201   delimiters .value_required:n = true ,
1202   small .bool_set:N = \l_@@_small_bool ,
1203   small .value_forbidden:n = true ,
1204   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1205   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1206   unknown .code:n =

```

```

1207 \@@_unknown_key:nn
1208 { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
1209 { Unknown~key~for~NiceMatrix }
1210 }

```

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

```

1211 \keys_define:nn { nicematrix / NiceTabular }
1212 {

```

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.

```

1213 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1214 \bool_set_true:N \l_@@_width_used_bool ,
1215 width .value_required:n = true ,
1216 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1217 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1218 tabularnote .value_required:n = true ,
1219 caption .tl_set:N = \l_@@_caption_tl ,
1220 caption .value_required:n = true ,
1221 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1222 short-caption .value_required:n = true ,
1223 label .tl_set:N = \l_@@_label_tl ,
1224 label .value_required:n = true ,
1225 last-col .code:n = \tl_if_empty:nF { #1 }
1226 { \@@_error:n { last-col~non~empty~for~NiceArray } }
1227 \int_zero:N \l_@@_last_col_int ,
1228 r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1229 l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1230 unknown .code:n =
1231 \@@_unknown_key:nn
1232 { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
1233 { Unknown~key~for~NiceTabular }
1234 }

```

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
```

```

1235 \keys_define:nn { nicematrix / CodeAfter }
1236 {
1237 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1238 delimiters / color .value_required:n = true ,
1239 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1240 rules .value_required:n = true ,
1241 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1242 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1243 sub-matrix .value_required:n = true ,
1244 unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1245 }

```

## 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}`).

```

1246 \cs_new_protected:Npn \@@_cell_begin:
1247 {

```

`\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.

```

1248 \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).

```

1249 \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.

```

1250 \cs_set_eq:NN \Hline \@@_Hline_in_cell:

```

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

```

1251 \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.

```

1252 \if_int_compare:w \c@jCol = \c_one_int
1253 \if_int_compare:w \l_@@_first_col_int = \c_one_int \@@_begin_of_row: \fi
1254 \fi:

```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` is in the `\@@_cell_end:`.

```

1255 \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1256 \@@_tuning_not_tabular_begin:
1257 \@@_tuning_first_row:
1258 \@@_tuning_last_row:
1259 \g_@@_row_style_tl
1260 }

```

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.

```

1261 \cs_new_protected:Npn \@@_tuning_first_row:
1262 {
1263   \if_int_compare:w \c@iRow = \c_zero_int
1264   \if_int_compare:w \c@jCol > \c_zero_int
1265   \l_@@_code_for_first_row_tl
1266   \xglobal \colorlet { nicematrix-first-row } { . }
1267   \fi:
1268   \fi:
1269 }

```

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.

```

1270 \cs_new_protected:Npn \@@_tuning_last_row:
1271 {
1272   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1273     \l_@@_code_for_last_row_tl
1274     \xglobal \colorlet { nicematrix-last-row } { . }
1275   \fi:
1276 }

```

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

```

1277 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1278 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1279 {
1280   \m@th
1281   $ % $

```

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

```

1282   \@@_tuning_key_small:
1283 }
1284 \cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $

```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```

1285 \cs_new_protected:Npn \@@_begin_of_row:
1286 {
1287   \int_gincr:N \c@iRow
1288   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1289   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1290   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1291   \pgfpicture
1292   \pgfrememberpicturepositiononpagetrue
1293   \pgfcoordinate
1294   { \@@_env: - row - \int_use:N \c@iRow - base }
1295   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1296   \str_if_empty:NF \l_@@_name_str
1297   {
1298     \pgfnodealias
1299     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1300     { \@@_env: - row - \int_use:N \c@iRow - base }
1301   }
1302   \endpgfpicture
1303 }

```

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.

```

1304 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1305 {

```

```

1306 \int_if_zero:nTF { \c@iRow }
1307 {
1308   \dim_compare:nNnT
1309     { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1310     { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1311   \dim_compare:nNnT
1312     { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1313     { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1314 }
1315 {
1316   \int_compare:nNnT { \c@iRow } = { \c_one_int }
1317   {
1318     \dim_compare:nNnT
1319       { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1320       { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1321   }
1322 }
1323 }
1324 \cs_new_protected:Npn \@@_rotate_cell_box:
1325 {
1326   \box_rotate:Nn \l_@@_cell_box { 90 }
1327   \bool_if:NTF \g_@@_rotate_c_bool
1328   {
1329     \hbox_set:Nn \l_@@_cell_box
1330     {
1331       \m@th
1332       $ % $
1333       \vcenter { \box_use:N \l_@@_cell_box }
1334       $ % $
1335     }
1336   }
1337   {
1338     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1339     {
1340       \vbox_set_top:Nn \l_@@_cell_box
1341       {
1342         \vbox_to_zero:n { }
1343         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1344         \box_use:N \l_@@_cell_box
1345       }
1346     }
1347   }
1348   \bool_gset_false:N \g_@@_rotate_bool
1349   \bool_gset_false:N \g_@@_rotate_c_bool
1350 }
1351 \cs_new_protected:Npn \@@_adjust_size_box:
1352 {
1353   \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1354   {
1355     \box_set_wd:Nn \l_@@_cell_box
1356     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1357     \dim_gzero:N \g_@@_blocks_wd_dim
1358   }
1359   \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1360   {
1361     \box_set_dp:Nn \l_@@_cell_box
1362     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1363     \dim_gzero:N \g_@@_blocks_dp_dim
1364   }
1365   \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1366   {
1367     \box_set_ht:Nn \l_@@_cell_box
1368     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }

```



```

1369         \dim_gzero:N \g_@@_blocks_ht_dim
1370     }
1371 }

1372 \cs_new_protected:Npn \@@_cell_end:
1373 {

```

The following command is nullified in the tabulars.

```

1374     \@@_tuning_not_tabular_end:
1375     \hbox_set_end:
1376     \@@_cell_end_i:
1377 }

1378 \cs_new_protected:Npn \@@_cell_end_i:
1379 {

```

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.

```

1380     \g_@@_cell_after_hook_tl
1381     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1382     \@@_adjust_size_box:

1383     \box_set_ht:Nn \l_@@_cell_box
1384     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1385     \box_set_dp:Nn \l_@@_cell_box
1386     { \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”).

```

1387     \@@_update_max_cell_width:

```

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

```

1388     \@@_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.

```

1389     \bool_if:NTF \g_@@_empty_cell_bool
1390     { \box_use_drop:N \l_@@_cell_box }
1391     {
1392         \bool_if:NTF \g_@@_not_empty_cell_bool
1393         { \@@_print_node_cell: }
1394         {
1395             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1396             { \@@_print_node_cell: }
1397             { \box_use_drop:N \l_@@_cell_box }
1398         }
1399     }

```

```

1400 \int_compare:nNt { \c@jCol } > { \g_@@_col_total_int }
1401 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1402 \bool_gset_false:N \g_@@_empty_cell_bool
1403 \bool_gset_false:N \g_@@_not_empty_cell_bool
1404 }

```

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

```

1405 \cs_new_protected:Npn \@@_update_max_cell_width:
1406 {
1407   \dim_gset:Nn \g_@@_max_cell_width_dim
1408   { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1409 }

```

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`).

```

1410 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1411 {
1412   \@@_math_toggle:
1413   \hbox_set_end:
1414   \bool_if:NF \g_@@_rotate_bool
1415   {
1416     \hbox_set:Nn \l_@@_cell_box
1417     {
1418       \makebox [ \l_@@_col_width_dim ] [ s ]
1419       { \hbox_unpack_drop:N \l_@@_cell_box }
1420     }
1421   }
1422   \@@_cell_end_i:
1423 }

```

```

1424 \pgfset
1425 {
1426   nicematrix / cell-node /.style =
1427   {
1428     inner~sep = \c_zero_dim ,
1429     minimum~width = \c_zero_dim
1430   }
1431 }

```

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.

```

1432 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1433 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1434 {
1435   \use:c
1436   {
1437     __siunitx_table_align_
1438     \bool_if:NTF \l__siunitx_table_text_bool
1439     { \l__siunitx_table_align_text_tl }
1440     { \l__siunitx_table_align_number_tl }
1441     :n
1442   }
1443   { #1 }
1444 }

```

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!

```

1445 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1446 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1447 {
1448   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1449   \hbox:n
1450   {
1451     \pgfsys@markposition
1452     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1453   }
1454   #1
1455   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1456   \hbox:n
1457   {
1458     \pgfsys@markposition
1459     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1460   }
1461 }

1462 \cs_new_protected:Npn \@@_print_node_cell:
1463 {
1464   \socket_use:nn { nicematrix / siunitx-wrap }
1465   { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1466 }

```

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

```

1467 \cs_new_protected:Npn \@@_node_cell:
1468 {
1469   \pgfpicture
1470   \pgfsetbaseline \c_zero_dim
1471   \pgfrememberpicturepositiononpagetrue
1472   \pgfset { nicematrix / cell-node }
1473   \pgfnode
1474   { rectangle }
1475   { base }
1476   {

```

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).

```

1477   \sys_if_engine_xetex:T { \set@color }
1478   \box_use:N \l_@@_cell_box
1479 }
1480 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1481 { \l_@@_pgf_node_code_tl }
1482 \str_if_empty:NF \l_@@_name_str
1483 {
1484   \pgfnodealias
1485   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1486   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1487 }
1488 \endpgfpicture
1489 }

```

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}

```

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

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).

```

1490 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1491 {
1492   \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
1493   { g_@@_ #2 _ lines _ tl }
1494   {
1495     \use:c { @@ _ draw _ #2 : nnn }
1496     { \int_use:N \c@iRow }
1497     { \int_use:N \c@jCol }
1498     { \exp_not:n { #3 } }
1499   }
1500 }

1501 \cs_new_protected:Npn \@@_array:n
1502 {
1503   \dim_set:Nn \col@sep
1504   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1505   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1506   { \def \@halignto { } }
1507   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1508   \@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)`.

```

1509   [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
1510   }
1511 \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`.

```

1512 \bool_if:NTF \c_@@_revtex_bool
1513 { \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`.

```

1514 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }

```

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

```

1515 \cs_new_protected:Npn \@@_create_row_node:
1516 {
1517   \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1518   {
1519     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1520     \@@_create_row_node_i:
1521   }
1522 }

```

```

1523 \cs_new_protected:Npn \@@_create_row_node_i:
1524 {

```

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

```

1525     \hbox
1526     {
1527         \bool_if:NT \l_@@_code_before_bool
1528         {
1529             \vtop
1530             {
1531                 \skip_vertical:N 0.5\arrayrulewidth
1532                 \pgfsys@markposition
1533                 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1534                 \skip_vertical:N -0.5\arrayrulewidth
1535             }
1536         }
1537         \pgfpicture
1538         \pgfrememberpicturepositiononpagetrue
1539         \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1540         { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1541         \str_if_empty:NF \l_@@_name_str
1542         {
1543             \pgfnodealias
1544             { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1545             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1546         }
1547         \endpgfpicture
1548     }
1549 }

```

```

1550 \cs_new_protected:Npn \@@_in_everycr:
1551 {
1552     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1553     \tbl_update_cell_data_for_next_row:
1554     \int_gzero:N \c@jCol
1555     \bool_gset_false:N \g_@@_after_col_zero_bool
1556     \bool_if:NF \g_@@_row_of_col_done_bool
1557     {
1558         \@@_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).

```

1559         \clist_if_empty:NF \l_@@_hlines_clist
1560         {
1561             \str_if_eq:eeF { \l_@@_hlines_clist } { all }
1562             {
1563                 \clist_if_in:NcT
1564                 \l_@@_hlines_clist
1565                 { \int_eval:n { \c@iRow + 1 } }
1566             }
1567         }

```

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.

```

1568             \int_compare:nNnT { \c@iRow } > { -1 }
1569             {
1570                 \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1571                 { \hrule height \arrayrulewidth width \c_zero_dim }
1572             }
1573         }
1574     }
1575 }
1576 }

```

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

```

1577 \cs_set_protected:Npn \@@_renew_dots:
1578 {
1579   \cs_set_eq:NN \ldots \@@_Ldots:
1580   \cs_set_eq:NN \cdots \@@_Cdots:
1581   \cs_set_eq:NN \vdots \@@_Vdots:
1582   \cs_set_eq:NN \ddots \@@_Ddots:
1583   \cs_set_eq:NN \iddots \@@_Iddots:
1584   \cs_set_eq:NN \dots \@@_Ldots:
1585   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1586 }

```

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

```

1587 \hook_gput_code:nnn { begindocument } { . }
1588 {
1589   \IfPackageLoadedTF { booktabs }
1590   {
1591     \cs_new_protected:Npn \@@_patch_booktabs:
1592     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1593   }
1594   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1595 }

```

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`.

```

1596 \cs_new_protected:Npn \@@_some_initialization:
1597 {
1598   \@@_everycr:
1599   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1600   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1601   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1602   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1603   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1604   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1605 }

```

`\@@_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`.

```

1606 \cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
1607 {

```

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

---

<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}`.

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

```
1608 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1609 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1610 \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.

```
1611 \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.

```
1612 \fp_gzero:N \g_@@_total_X_weight_fp
1613 \bool_gset_false:N \g_@@_V_of_X_bool
1614 \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
1615 \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
1616 \@@_patch_booktabs:
1617 \box_clear_new:N \l_@@_cell_box
1618 \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}`).

```
1619 \bool_if:NT \l_@@_small_bool
1620 {
1621   \def \arraystretch { 0.47 }
1622   \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```
1623 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1624 }
```

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!

```
1625 \bool_if:NT \g_@@_create_cell_nodes_bool
1626 {
1627   \tl_put_right:Nn \@@_begin_of_row:
1628   {
1629     \pgfsys@markposition
1630     { \@@_env: - row - \int_use:N \c@iRow - base }
1631   }
1632   \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
1633 }
```

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

```
1634 \bool_if:NF \c_@@_revtex_bool
1635 {
1636   \def \ar@ialign
1637   {
1638     \tbl_init_cell_data_for_table:
1639     \@@_some_initialization:
1640     \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`.

```

1641         \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
1642         \halign
1643     }
1644 }

```

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).

```

1645     \bool_if:NT \c_@@_revtex_bool
1646     {
1647         \IfPackageLoadedT { colortbl }
1648         { \cs_set_protected:Npn \CT@setup { } }
1649     }

```

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).

```

1650     \cs_set_eq:NN \@@_old_ldots: \ldots
1651     \cs_set_eq:NN \@@_old_cdots: \cdots
1652     \cs_set_eq:NN \@@_old_vdots: \vdots
1653     \cs_set_eq:NN \@@_old_ddots: \ddots
1654     \cs_set_eq:NN \@@_old_iddots: \iddots
1655     \bool_if:NTF \l_@@_standard_cline_bool
1656     { \cs_set_eq:NN \cline \@@_standard_cline: }
1657     { \cs_set_eq:NN \cline \@@_cline: }
1658     \cs_set_eq:NN \Ldots \@@_Ldots:
1659     \cs_set_eq:NN \Cdots \@@_Cdots:
1660     \cs_set_eq:NN \Vdots \@@_Vdots:
1661     \cs_set_eq:NN \Ddots \@@_Ddots:
1662     \cs_set_eq:NN \Iddots \@@_Iddots:
1663     \cs_set_eq:NN \Hline \@@_Hline:
1664     \cs_set_eq:NN \Hspace \@@_Hspace:
1665     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1666     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1667     \cs_set_eq:NN \Block \@@_Block:
1668     \cs_set_eq:NN \rotate \@@_rotate:
1669     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1670     \cs_set_eq:NN \dotfill \@@_dotfill:
1671     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1672     \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
1673     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1674     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1675     \cs_set_eq:NN \TopRule \@@_TopRule
1676     \cs_set_eq:NN \MidRule \@@_MidRule
1677     \cs_set_eq:NN \BottomRule \@@_BottomRule
1678     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1679     \cs_set_eq:NN \Hbrace \@@_Hbrace
1680     \cs_set_eq:NN \Vbrace \@@_Vbrace
1681     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1682     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1683     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1684     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1685     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1686     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1687     \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1688     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1689     \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1690     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }

```



```

1691 \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:.`

```

1692 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1693 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1694 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1695 \@@_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!).

```

1696 \tl_if_exist:NT \l_@@_note_in_caption_tl
1697 {
1698   \tl_if_empty:NF \l_@@_note_in_caption_tl
1699   {
1700     \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
1701     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1702   }
1703 }

```

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).

```

1704 \seq_gclear:N \g_@@_multicolumn_cells_seq
1705 \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).

```

1706 \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`).

```

1707 \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.

```

1708 \int_gzero:N \g_@@_col_total_int
1709 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1710 \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.

```

1711 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1712 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1713 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1714 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1715 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1716 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1717 \tl_gclear:N \g_nicematrix_code_before_tl
1718 \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]`).

```

1719 \dim_zero_new:N \l_@@_left_delim_dim
1720 \dim_zero_new:N \l_@@_right_delim_dim
1721 \bool_if:NTF \g_@@_delims_bool
1722 {

```

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

```

1723 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1724 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1725 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1726 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1727 }
1728 {
1729 \dim_gset:Nn \l_@@_left_delim_dim
1730 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1731 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1732 }
1733 }

```

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.

```

1734 \cs_new_protected:Npn \@@_pre_array:
1735 {
1736 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1737 \int_gzero_new:N \c@iRow
1738 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1739 \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).

```

1740 \int_compare:nNnT \l_@@_last_row_int > 0
1741 { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
1742 \int_compare:nNnT \l_@@_last_col_int > 0
1743 { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
1744 \bool_if:NT \g_@@_aux_found_bool
1745 {
1746 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
1747 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
1748 \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1749 \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
1750 }

```

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).

```

1751 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
1752 {
1753 \bool_set_true:N \l_@@_last_row_without_value_bool
1754 \bool_if:NT \g_@@_aux_found_bool
1755 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1756 }
1757 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1758 {
1759 \bool_if:NT \g_@@_aux_found_bool

```

```

1760      { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1761    }

```

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”.

```

1762    \int_compare:nNtT { \l_@@_last_row_int } > { -2 }
1763    {
1764      \tl_put_right:Nn \@@_update_for_first_and_last_row:
1765      {
1766        \dim_compare:nNtT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1767        { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1768        \dim_compare:nNtT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1769        { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1770      }
1771    }

1772    \seq_gclear:N \g_@@_cols_vlism_seq
1773    \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1774    \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

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

```

1775    \@@_pre_array_after_CodeBefore:

```

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

```

1776    \hbox_set:Nw \l_@@_the_array_box
1777    \skip_horizontal:N \l_@@_left_margin_dim
1778    \skip_horizontal:N \l_@@_extra_left_margin_dim
1779    \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).

```

1780    \m@th
1781    $ % $
1782    \bool_if:NtF \l_@@_light_syntax_bool
1783    { \use:c { @@-light-syntax } }
1784    { \use:c { @@-normal-syntax } }
1785  }

```

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

```

1786    \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1787    {
1788      \tl_set:Nn \l_tmpa_tl { #1 }
1789      \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1790      { \@@_rescan_for_spanish:N \l_tmpa_tl }
1791      \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1792      \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...

```

1793    \@@_pre_array:
1794  }

```

## 9 The \CodeBefore

```
1795 \cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
```

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).

```
1796 \cs_new_protected:Npn \@@_pre_code_before:
1797 {
```

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.

```
1798 \pgfsys@markposition { \@@_env: - position }
1799 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1800 \pgfpicture
1801 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1802 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1803 {
1804   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1805   \pgfcoordinate { \@@_env: - row - ##1 }
1806   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1807 }
```

Now, the recreation of the col nodes.

```
1808 \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1809 {
1810   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1811   \pgfcoordinate { \@@_env: - col - ##1 }
1812   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1813 }
```

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

```
1814 \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1815 \endpgfpicture
```

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

```
1816 \@@_create_diag_nodes:
```

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

```
1817 \@@_create_blocks_nodes:
1818 \IfPackageLoadedT { tikz }
1819 {
1820   \tikzset
1821   {
1822     every-picture / .style =
1823     { overlay , name~prefix = \@@_env: - }
1824   }
1825 }
1826 \cs_set_eq:NN \cellcolor \@@_cellcolor
1827 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1828 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1829 \cs_set_eq:NN \rowcolor \@@_rowcolor
1830 \cs_set_eq:NN \rowcolors \@@_rowcolors
1831 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1832 \cs_set_eq:NN \arraycolor \@@_arraycolor
1833 \cs_set_eq:NN \columncolor \@@_columncolor
1834 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1835 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1836 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1837 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1838 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1839 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1840 }
```

```

1841 \cs_new_protected:Npn \@@_exec_code_before:
1842 {

```

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...

```

1843 \clist_map_inline:Nn \l_@@_corners_cells_clist
1844 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1845 \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`.

```

1846 \@@_add_to_colors_seq:nn { { nocolor } } { }
1847 \bool_gset_false:N \g_@@_create_cell_nodes_bool
1848 \group_begin:

```

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

```

1849 \if_mode_math:
1850 \@@_exec_code_before_i:
1851 \else:
1852 $ % $
1853 \@@_exec_code_before_i:
1854 $ % $
1855 \fi:
1856 \group_end:
1857 }

```

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`).

```

1858 \cs_new_protected:Npn \@@_exec_code_before_i:
1859 {
1860 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1861 { \@@_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`.

```

1862 \exp_last_unbraced:No \@@_CodeBefore_keys:
1863 \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.

```

1864 \@@_actually_color:
1865 \l_@@_code_before_tl
1866 \q_stop
1867 }

```

```

1868 \keys_define:nn { nicematrix / CodeBefore }
1869 {
1870 create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1871 create-cell-nodes .default:n = true ,
1872 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1873 sub-matrix .value_required:n = true ,
1874 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1875 delimiters / color .value_required:n = true ,
1876 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1877 }

1878 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1879 {

```

```

1880 \keys_set:nn { nicematrix / CodeBefore } { #1 }
1881 \@@_CodeBefore:w
1882 }

```

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.

```

1883 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1884 {
1885   \bool_if:NTF \g_@@_aux_found_bool
1886   {
1887     \@@_pre_code_before:
1888     \legacy_if:nF { measuring@ } { #1 }
1889   }

```

If we are in the first compilation, you won't really execute the `\CodeBefore` but we have to execute some instructions of creation of PGF/TikZ pictures in order to have the correct `aux` file in the next run (hence, we avoid to "lose" a run).

```

1890 {
1891   \pgfsys@markposition { \@@_env: - position }
1892   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1893   \pgfpicture
1894   \pgf@relevantforpicturesizefalse
1895   \endpgfpicture

```

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

```

1896   \pgfpicture
1897   \pgfrememberpicturepositiononpagetrue
1898   \endpgfpicture

```

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

```

1899   \pgfpicture
1900   \pgf@relevantforpicturesizefalse
1901   \pgfrememberpicturepositiononpagetrue
1902   \endpgfpicture

```

The following picture corresponds `\@@_actually_color:`

```

1903   \pgfpicture
1904   \pgf@relevantforpicturesizefalse
1905   \endpgfpicture
1906 }
1907 }

```

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.

```

1908 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1909 {
1910   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1911   {
1912     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1913     \pgfcoordinate { \@@_env: - row - ##1 - base }
1914     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1915     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1916     {
1917       \cs_if_exist:cT
1918       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1919       {
1920         \pgfsys@getposition
1921         { \@@_env: - ##1 - #####1 - NW }
1922         \@@_node_position:
1923         \pgfsys@getposition
1924         { \@@_env: - ##1 - #####1 - SE }
1925         \@@_node_position_i:

```

```

1926         \@@_pgf_rect_node:nnn
1927         { \@@_env: - ##1 - ###1 }
1928         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1929         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1930     }
1931 }
1932 }
1933 \@@_create_extra_nodes:
1934 \@@_create_aliases_last:
1935 }

1936 \cs_new_protected:Npn \@@_create_aliases_last:
1937 {
1938     \int_step_inline:nn { \c@iRow }
1939     {
1940         \pgfnodealias
1941         { \@@_env: - ##1 - last }
1942         { \@@_env: - ##1 - \int_use:N \c@jCol }
1943     }
1944     \int_step_inline:nn { \c@jCol }
1945     {
1946         \pgfnodealias
1947         { \@@_env: - last - ##1 }
1948         { \@@_env: - \int_use:N \c@iRow - ##1 }
1949     }
1950     \pgfnodealias
1951     { \@@_env: - last - last }
1952     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1953 }

1954 \cs_new_protected:Npn \@@_create_blocks_nodes:
1955 {
1956     \pgfpicture
1957     \pgf@relevantforpicturesizefalse
1958     \pgfrememberpicturepositiononpagetrue
1959     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1960     { \@@_create_one_block_node:nnnnn ##1 }
1961     \endpgfpicture
1962 }

```

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>

```

1963 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1964 {
1965     \tl_if_empty:nF { #5 }
1966     {
1967         \@@_qpoint:n { col - #2 }
1968         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1969         \@@_qpoint:n { #1 }
1970         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1971         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1972         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1973         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1974         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1975         \@@_pgf_rect_node:nnnnn
1976         { \@@_env: - #5 }
1977         { \dim_use:N \l_tmpa_dim }
1978         { \dim_use:N \l_tmpb_dim }
1979         { \dim_use:N \l_@@_tmpc_dim }

```

---

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

```

1980         { \dim_use:N \l_@@_tmpd_dim }
1981     }
1982 }

1983 \cs_new_protected:Npn \@@_patch_for_revtext:
1984 {
1985     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1986     \cs_set_eq:NN \@array \@array@array
1987     \cs_set_eq:NN \@tabular \@tabular@array
1988     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1989     \cs_set_eq:NN \array \array@array
1990     \cs_set_eq:NN \endarray \endarray@array
1991     \cs_set:Npn \endtabular { \endarray $\egroup } % $
1992     \cs_set_eq:NN \@mkpream \@mkpream@array
1993     \cs_set_eq:NN \@classx \@classx@array
1994     \cs_set_eq:NN \insert@column \insert@column@array
1995     \cs_set_eq:NN \@arraycr \@arraycr@array
1996     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1997     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1998 }

```

## 10 The environment `{NiceArrayWithDelims}`

```

1999 \NewDocumentEnvironment { NiceArrayWithDelims }
2000 { m m O { } m ! O { } t \CodeBefore }
2001 {
2002     \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtext: }
2003
2004     \@@_provide_pgfsyspdfmark:
2005     \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.

```

2005     \bgroup

2006     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2007     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2008     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
2009     \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

2010     \int_gzero:N \g_@@_block_box_int
2011     \dim_gzero:N \g_@@_width_last_col_dim
2012     \dim_gzero:N \g_@@_width_first_col_dim
2013     \bool_gset_false:N \g_@@_row_of_col_done_bool
2014     \str_if_empty:NT \g_@@_name_env_str
2015     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
2016     \bool_if:NTF \l_@@_tabular_bool
2017     { \mode_leave_vertical: }
2018     { \@@_test_if_math_mode: }
2019     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
2020     \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.

```

2021     \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

---

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



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.

```

2022 \cs_if_exist:NT \tikz@library@external@loaded
2023 {
2024     \tikzexternaldisable
2025     \cs_if_exist:NT \ifstandalone
2026     { \tikzset { external / optimize = false } }
2027 }

```

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

```

2028 \int_gincr:N \g_@@_env_int
2029 \bool_if:NF \l_@@_block_auto_columns_width_bool
2030 { \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.

```

2031 \seq_gclear:N \g_@@_blocks_seq
2032 \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`.

```

2033 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
2034 \seq_gclear:N \g_@@_pos_of_xdots_seq
2035 \tl_gclear_new:N \g_@@_code_before_tl
2036 \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.

```

2037 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2038 {
2039     \bool_gset_true:N \g_@@_aux_found_bool
2040     \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
2041 }
2042 { \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.

```

2043 \tl_gclear:N \g_@@_aux_tl
2044 \tl_if_empty:NF \g_@@_code_before_tl
2045 {
2046     \bool_set_true:N \l_@@_code_before_bool
2047     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
2048 }
2049 \tl_if_empty:NF \g_@@_pre_code_before_tl
2050 { \bool_set_true:N \l_@@_code_before_bool }

```

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

```

2051 \bool_if:NTF \g_@@_delims_bool
2052 { \keys_set:nn { nicematrix / pNiceArray } }
2053 { \keys_set:nn { nicematrix / NiceArray } }
2054 { #3 , #5 }

2055 \@@_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:.`

```

2056 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }

```

2057 }

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

```
2058 {
2059   \bool_if:NTF \l_@@_light_syntax_bool
2060     { \use:c { end @@-light-syntax } }
2061     { \use:c { end @@-normal-syntax } }
2062   $ % $
2063   \skip_horizontal:N \l_@@_right_margin_dim
2064   \skip_horizontal:N \l_@@_extra_right_margin_dim
2065   \hbox_set_end:
2066   \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.

```
2067   \bool_if:NT \l_@@_width_used_bool
2068   {
2069     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2070     { \@@_error_or_warning:n { width~without~X~columns } }
2071   }
```

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  $x$ .

```
2072   \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2073     { \@@_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).

```
2074   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2075   {
2076     \bool_if:NF \l_@@_last_row_without_value_bool
2077     {
2078       \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2079       {
2080         \@@_error:n { Wrong~last~row }
2081         \int_set_eq:NN \l_@@_last_row_int \c@iRow
2082       }
2083     }
2084   }
```

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

```
2085   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2086   \bool_if:NTF \g_@@_last_col_found_bool
2087     { \int_gdecr:N \c@jCol }
2088     {
2089       \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2090       { \@@_error:n { last~col~not~used } }
2091     }
```

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

```
2092   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2093   \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2094     { \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. 93).

---

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

```

2095 \int_if_zero:nT { \l_@@_first_col_int }
2096 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2097 \bool_if:nTF { ! \g_@@_delims_bool }
2098 {
2099   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
2100   { \@@_use_arraybox_with_notes_c: }
2101   {
2102     \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
2103     { \@@_use_arraybox_with_notes_b: }
2104     { \@@_use_arraybox_with_notes: }
2105   }
2106 }

```

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).

```

2107 {
2108   \int_if_zero:nTF { \l_@@_first_row_int }
2109   {
2110     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2111     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2112   }
2113   { \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>

```

2114 \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2115 {
2116   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2117   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2118 }
2119 { \dim_zero:N \l_tmpb_dim }
2120 \hbox_set:Nn \l_tmpa_box
2121 {
2122   \m@th
2123   $ % $
2124   \@@_color:o \l_@@_delimiters_color_tl
2125   \exp_after:wN \left \g_@@_left_delim_tl
2126   \vcenter
2127   {

```

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.

```

2128   \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2129   \hbox
2130   {
2131     \bool_if:NTF \l_@@_tabular_bool
2132     { \skip_horizontal:n { - \tabcolsep } }
2133     { \skip_horizontal:n { - \arraycolsep } }
2134     \@@_use_arraybox_with_notes_c:
2135     \bool_if:NTF \l_@@_tabular_bool
2136     { \skip_horizontal:n { - \tabcolsep } }
2137     { \skip_horizontal:n { - \arraycolsep } }
2138   }

```

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

```

2139   \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2140 }
2141 \exp_after:wN \right \g_@@_right_delim_tl
2142 $ % $
2143 }

```

---

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

Now, the box `\l_tmpa_box` is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

2144     \bool_if:NTF \l_@@_delimiters_max_width_bool
2145     {
2146         \@@_put_box_in_flow_bis:nn
2147         { \g_@@_left_delim_tl }
2148         { \g_@@_right_delim_tl }
2149     }
2150     \@@_put_box_in_flow:
2151 }

```

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. 94).

```

2152     \bool_if:NT \g_@@_last_col_found_bool
2153     { \skip_horizontal:N \g_@@_width_last_col_dim }
2154     \bool_if:NT \l_@@_preamble_bool
2155     {
2156         \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2157         { \@@_err_columns_not_used: }
2158     }
2159     \@@_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.

```

2160     \egroup

```

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

```

2161     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2162     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2163     \iow_now:Ne \@mainaux
2164     {
2165         \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2166         \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2167         { \exp_not:o \g_@@_aux_tl }
2168     }
2169     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2170     \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2171 }

```

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

```

2172 \cs_new_protected:Npn \@@_err_columns_not_used:
2173 {
2174     \@@_warning:n { columns~not~used }
2175     \cs_gset:Npn \@@_err_columns_not_used: { }
2176 }

```

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  $x$ .

```

2177 \cs_new_protected:Npn \@@_compute_width_X:
2178 {
2179     \tl_gput_right:Ne \g_@@_aux_tl
2180     {
2181         \bool_set_true:N \l_@@_X_columns_aux_bool
2182         \dim_set:Nn \l_@@_X_columns_dim
2183         {

```

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`

```

2184 \bool_lazy_and:nnTF
2185 { \g_@@_V_of_X_bool }
2186 { \l_@@_X_columns_aux_bool }
2187 { \dim_use:N \l_@@_X_columns_dim }
2188 {
2189   \dim_compare:nNnTF
2190   {
2191     \dim_abs:n
2192     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2193   }
2194   <
2195   { 0.001 pt }
2196   { \dim_use:N \l_@@_X_columns_dim }
2197   {
2198     \dim_eval:n
2199     {
2200       \l_@@_X_columns_dim
2201       +
2202       \fp_to_dim:n
2203       {
2204         (
2205           \dim_eval:n
2206           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2207         )
2208         / \fp_use:N \g_@@_total_X_weight_fp
2209       }
2210     }
2211   }
2212 }
2213 }
2214 }
2215 }

```

## 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`.

```

2216 \cs_new_protected:Npn \@@_transform_preamble:
2217 {
2218   \@@_transform_preamble_i:
2219   \@@_transform_preamble_ii:
2220 }
2221 \cs_new_protected:Npn \@@_transform_preamble_i:
2222 {
2223   \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`).

```

2224 \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.

```

2225 \bool_gset_false:N \g_tmpb_bool

```

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

```

2226 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2227 \int_zero:N \l_tmpa_int
2228 \tl_gclear:N \g_@@_array_preamble_tl
2229 \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2230 {
2231   \tl_gset:Nn \g_@@_array_preamble_tl
2232     { ! { \skip_horizontal:N \arrayrulewidth } }
2233 }
2234 {
2235   \clist_if_in:NnT \l_@@_vlines_clist 1
2236   {
2237     \tl_gset:Nn \g_@@_array_preamble_tl
2238       { ! { \skip_horizontal:N \arrayrulewidth } }
2239   }
2240 }

```

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

```

2241 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2242 \int_gset:eq:NN \g_@@_static_num_of_col_int \c@jCol

2243 \@@_replace_columncolor:
2244 }

2245 \cs_new_protected:Npn \@@_transform_preamble_ii:
2246 {

```

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

```

2247 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2248 {
2249   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2250   { \bool_gset_true:N \g_@@_delims_bool }
2251 }
2252 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2253 \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).

```

2254 \int_if_zero:nTF { \l_@@_first_col_int }
2255 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2256 {
2257   \bool_if:NF \g_@@_delims_bool
2258   {
2259     \bool_if:NF \l_@@_tabular_bool
2260     {
2261       \clist_if_empty:NT \l_@@_vlines_clist
2262       {
2263         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2264         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2265       }
2266     }
2267   }
2268 }
2269 \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2270 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2271 {
2272   \bool_if:NF \g_@@_delims_bool
2273   {

```

```

2274         \bool_if:NF \l_@@_tabular_bool
2275         {
2276             \clist_if_empty:NT \l_@@_vlines_clist
2277             {
2278                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2279                 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2280             }
2281         }
2282     }
2283 }

```

We try to give a good error message when the final user puts more columns than allowed by the preamble of the array. The mechanism consists of an extra column. However, if tagging is in force, that dummy extra column will be tagged (with <TD> tags) and that's why we disable that mechanism when tagging is in force.

```

2284     \tag_if_active:F
2285     {

```

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

```

2286         \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2287         {
2288             \tl_gput_right:Nn \g_@@_array_preamble_tl
2289             { > { \@@_err_too_many_cols: } 1 }
2290         }
2291     }
2292 }

```

We have used to add a last column to raise a good error message when the user puts more columns than allowed by its preamble. For technical reasons, it was not possible to do that in {NiceTabular\*} and that's why we used to control that with the value of \l\_@@\_tabular\_width\_dim).

The preamble provided by the final user will be read by a finite automata. The following function \@@\_rec\_preamble:n will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```

2293 \cs_new_protected:Npn \@@_rec_preamble:n #1
2294 {

```

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>

```

2295     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2296     { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2297     {

```

Now, the columns defined by \newcolumntype of array.

```

2298         \cs_if_exist:cTF { NC @ find @ #1 }
2299         {
2300             \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2301             \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2302         }
2303         {
2304             \str_if_eq:nnTF { #1 } { S }
2305             { \@@_fatal:n { unknown~column~type~S } }
2306             { \@@_fatal:nn { unknown~column~type } { #1 } }
2307         }
2308     }
2309 }

```

---

<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.

For c, l and r

```

2310 \cs_new_protected:Npn \@@_c: #1
2311 {
2312   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2313   \tl_gclear:N \g_@@_pre_cell_tl
2314   \tl_gput_right:Nn \g_@@_array_preamble_tl
2315     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2316   \int_gincr:N \c@jCol
2317   \@@_rec_preamble_after_col:n
2318 }

2319 \cs_new_protected:Npn \@@_l: #1
2320 {
2321   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2322   \tl_gclear:N \g_@@_pre_cell_tl
2323   \tl_gput_right:Nn \g_@@_array_preamble_tl
2324     {
2325       > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2326       l
2327       < \@@_cell_end:
2328     }
2329   \int_gincr:N \c@jCol
2330   \@@_rec_preamble_after_col:n
2331 }

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

```

For ! and @

```

2345 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2346 {
2347   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2348   \@@_rec_preamble:n
2349 }
2350 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2351 \cs_new_protected:cpn { @@ _ | : } #1
2352 {

```

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

```

2353   \int_incr:N \l_tmpa_int
2354   \@@_make_preamble_i_i:n
2355 }

2356 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2357 {

```

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

```

2358   \str_if_eq:nnTF { #1 } { | }
2359     { \use:c { @@ _ | : } | }
2360     { \@@_make_preamble_i_ii:nn { } #1 }
2361 }

```



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

```

2362 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2363 {
2364   \str_if_eq:nnTF { #2 } { [ ] }
2365   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2366   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2367 }
2368 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2369 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2370 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2371 {
2372   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2373   \tl_gput_right:Ne \g_@@_array_preamble_tl
2374   {

```

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

```

2375     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2376   }
2377   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2378   {
2379     \@@_vline:n
2380     {
2381       position = \int_eval:n { \c@jCol + 1 } ,
2382       multiplicity = \int_use:N \l_tmpa_int ,
2383       total-width = \dim_use:N \l_@@_rule_width_dim ,
2384       #2
2385     }

```

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.

```

2386   }
2387   \int_zero:N \l_tmpa_int
2388   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2389   \@@_rec_preamble:n #1
2390 }

2391 \cs_new_protected:cpn { @@ _ > : } #1 #2
2392 {
2393   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2394   \@@_rec_preamble:n
2395 }

2396 \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.

```

2397 \keys_define:nn { nicematrix / p-column }
2398 {
2399   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2400   r .value_forbidden:n = true ,
2401   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2402   c .value_forbidden:n = true ,
2403   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2404   l .value_forbidden:n = true ,
2405   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2406   S .value_forbidden:n = true ,
2407   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2408   p .value_forbidden:n = true ,
2409   t .meta:n = p ,
2410   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2411   m .value_forbidden:n = true ,

```

```

2412     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2413     b .value_forbidden:n = true
2414 }

```

For p but also b and m.

```

2415 \cs_new_protected:Npn \@@_p: #1
2416 {
2417     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2418     \@@_make_preamble_ii_i:n
2419 }
2420 \cs_set_eq:NN \@@_b: \@@_p:
2421 \cs_set_eq:NN \@@_m: \@@_p:
2422 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2423 {
2424     \str_if_eq:nnTF { #1 } { [ ]
2425     { \@@_make_preamble_ii_ii:w [ ]
2426       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2427     }
2428 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2429 { \@@_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.

```

2430 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2431 {

```

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).

```

2432     \str_set:Nn \l_@@_hpos_col_str { j }
2433     \@@_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`.

```

2434     \setlength { \l_tmpa_dim } { #2 }
2435     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2436 }
2437 \cs_new_protected:Npn \@@_keys_p_column:n #1
2438 { \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.

```

2439 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2440 {

```

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

```

2441     \use:e
2442     {
2443         \@@_make_preamble_ii_vi:nnnnnnnn
2444         { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
2445         { #1 }
2446     }

```

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.

```

2447     \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
2448     { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2449     {

```

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

```

2450         \def \exp_not:N \l_@@_hpos_cell_tl
2451         { \str_lowercase:f { \l_@@_hpos_col_str } }
2452     }
2453     \IfPackageLoadedTF { ragged2e }
2454     {
2455         \str_case:on \l_@@_hpos_col_str
2456         {

```

The following `\exp_not:N` are mandatory.

```

2457         c { \exp_not:N \Centering }
2458         l { \exp_not:N \RaggedRight }
2459         r { \exp_not:N \RaggedLeft }
2460     }
2461 }
2462 {
2463     \str_case:on \l_@@_hpos_col_str
2464     {
2465         c { \exp_not:N \centering }
2466         l { \exp_not:N \raggedright }
2467         r { \exp_not:N \raggedleft }
2468     }
2469 }
2470 #3
2471 }
2472 { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
2473 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
2474 { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
2475 { #2 }
2476 {
2477     \str_case:onF \l_@@_hpos_col_str
2478     {
2479         { j } { c }
2480         { si } { c }
2481     }

```

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

```

2482         { \str_lowercase:f \l_@@_hpos_col_str }
2483     }
2484 }

```

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

```

2485     \int_gincr:N \c@jCol
2486     \@@_rec_preamble_after_col:n
2487 }

```

**#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*.

```

2488 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2489 {
2490     \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
2491     {

```

```

2492     \tl_gput_right:Nn \g_@@_array_preamble_tl
2493     { > \@@_test_if_empty_for_S: }
2494   }
2495   {
2496     \str_if_eq:eeTF { #7 } { varwidth }
2497     {
2498       \tl_gput_right:Nn \g_@@_array_preamble_tl
2499       { > \@@_test_if_empty_varwidth: }
2500     }
2501     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2502   }
2503   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2504   \tl_gclear:N \g_@@_pre_cell_tl
2505   \tl_gput_right:Nn \g_@@_array_preamble_tl
2506   {
2507     > {

```

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.

```

2508     \dim_set:Nn \l_@@_col_width_dim { #2 }
2509     \@@_cell_begin:

```

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

```

2510     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2511     \everypar
2512     {
2513       \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2514       \everypar { }
2515     }

```

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

```

2516     #3

```

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

```

2517     \g_@@_row_style_tl
2518     \arraybackslash
2519     #5
2520   }
2521   #8
2522   < {
2523     #6

```

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

```

2524     \@finalstrut \@arstrutbox
2525     \use:c { end #7 }

```

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

```

2526     #4
2527     \@@_cell_end:
2528   }
2529 }
2530 }

```

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

```

2531 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2532 {

```

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 `&`).

```

2533 \group_align_safe_begin:
2534 \peek_meaning:NTF &
2535 { \@@_the_cell_is_empty: }
2536 {
2537   \peek_meaning:NTF \\\
2538   { \@@_the_cell_is_empty: }
2539   {
2540     \peek_meaning:NTF \crcr
2541     \@@_the_cell_is_empty:
2542     \group_align_safe_end:
2543   }
2544 }
2545 }

```

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

```

2546 \cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
2547 {
2548   \group_align_safe_begin:
2549   \peek_meaning:NTF &
2550   { \@@_the_cell_is_empty_varwidth: }
2551   {
2552     \peek_meaning:NTF \\\
2553     { \@@_the_cell_is_empty_varwidth: }
2554     {
2555       \peek_meaning:NTF \crcr
2556       \@@_the_cell_is_empty_varwidth:
2557       \group_align_safe_end:
2558     }
2559   }
2560 }
2561 \cs_new_protected:Npn \@@_the_cell_is_empty:
2562 {
2563   \group_align_safe_end:
2564   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2565   {

```

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

```

2566 \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```

If all the cells of the column are empty, we still must have a column with the width required by the column of type p (or b, or m).

```

2567 \skip_horizontal:N \l_@@_col_width_dim
2568 }
2569 }
2570 \cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
2571 {
2572   \group_align_safe_end:
2573   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2574   { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2575 }
2576 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2577 {
2578   \peek_meaning:NT \__siunitx_table_skip:n
2579   { \bool_gset_true:N \g_@@_empty_cell_bool }
2580 }

```

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.

```
2581 \cs_new_protected:Npn \@@_center_cell_box:
2582 {
```

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

```
2583 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2584 {
2585     \dim_compare:nNnT
2586     { \box_ht:N \l_@@_cell_box }
2587     >
```

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).

```
2588     { \box_ht:N \strutbox }
2589     {
2590         \hbox_set:Nn \l_@@_cell_box
2591         {
2592             \box_move_down:nn
2593             {
2594                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2595                   + \baselineskip ) / 2
2596             }
2597             { \box_use:N \l_@@_cell_box }
2598         }
2599     }
2600 }
2601 }
```

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

```
2602 \cs_new_protected:Npn \@@_V: #1 #2
2603 {
2604     \str_if_eq:nnTF { #2 } { [ ] }
2605     { \@@_make_preamble_V_i:w [ ] }
2606     { \@@_make_preamble_V_i:w [ ] { #2 } }
2607 }
2608 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2609 { \@@_make_preamble_V_ii:nn { #1 } }
2610 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2611 {
2612     \str_set:Nn \l_@@_vpos_col_str { p }
2613     \str_set:Nn \l_@@_hpos_col_str { j }
2614     \@@_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`.

```
2615 \setlength { \l_tmpa_dim } { #2 }
2616 \IfPackageLoadedTF { varwidth }
2617 { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2618 {
2619     \@@_error_or_warning:n { varwidth-not-loaded }
2620     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2621 }
2622 }
```

For `w` and `W`

```
2623 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2624 \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.

```

2625 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2626 {
2627   \str_if_eq:nnTF { #3 } { s }
2628     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2629     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2630 }

```

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.

```

2631 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2632 {
2633   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2634   \tl_gclear:N \g_@@_pre_cell_tl
2635   \tl_gput_right:Nn \g_@@_array_preamble_tl
2636     {
2637       > {

```

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.

```

2638       \setlength { \l_@@_col_width_dim } { #2 }
2639       \@@_cell_begin:
2640       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2641     }
2642     c
2643     < {
2644       \@@_cell_end_for_w_s:
2645       #1
2646       \@@_adjust_size_box:
2647       \box_use_drop:N \l_@@_cell_box
2648     }
2649   }
2650   \int_gincr:N \c@jCol
2651   \@@_rec_preamble_after_col:n
2652 }

```

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

```

2653 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2654 {
2655   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2656   \tl_gclear:N \g_@@_pre_cell_tl
2657   \tl_gput_right:Nn \g_@@_array_preamble_tl
2658     {
2659       > {

```

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.

```

2660       \setlength { \l_@@_col_width_dim } { #4 }
2661       \hbox_set:Nw \l_@@_cell_box
2662       \@@_cell_begin:
2663       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2664     }
2665     c

```

```

2666         < {
2667             \@@_cell_end:
2668             \hbox_set_end:
2669             #1
2670             \@@_adjust_size_box:
2671             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2672         }
2673     }

```

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

```

2674     \int_gincr:N \c@jCol
2675     \@@_rec_preamble_after_col:n
2676 }

2677 \cs_new_protected:Npn \@@_special_W:
2678 {
2679     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2680     { \@@_warning:n { W~warning } }
2681 }

```

For S (of siunitx).

```

2682 \cs_new_protected:Npn \@@_S: #1 #2
2683 {
2684     \str_if_eq:nnTF { #2 } { [ ] }
2685     { \@@_make_preamble_S:w [ ] }
2686     { \@@_make_preamble_S:w [ ] { #2 } }
2687 }

2688 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2689 { \@@_make_preamble_S:i:n { #1 } }

2690 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2691 {
2692     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2693     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2694     \tl_gc_clear:N \g_@@_pre_cell_tl
2695     \tl_gput_right:Nn \g_@@_array_preamble_tl
2696     {
2697         > {

```

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).

```

2698         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2699         \keys_set:nn { siunitx } { #1 }
2700         \@@_cell_begin:
2701         \siunitx_cell_begin:w
2702     }
2703     c
2704     <
2705     {
2706         \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 it will stay local within the cell of the underlying \halign).

```

2707         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2708         {
2709             \bool_if:NTF \l__siunitx_table_text_bool
2710             { \bool_set_true:N }
2711             { \bool_set_false:N }
2712             \l__siunitx_table_text_bool

```



```

2713     }
2714     \@@_cell_end:
2715 }
2716 }

```

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

```

2717     \int_gincr:N \c@jCol
2718     \@@_rec_preamble_after_col:n
2719 }

```

For (, [ and \{.

```

2720 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2721 {
2722     \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.

```

2723     \int_if_zero:nTF { \c@jCol }
2724     {
2725         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2726         {

```

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

```

2727             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2728             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2729             \@@_rec_preamble:n #2
2730         }
2731     {
2732         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2733         \@@_make_preamble_iv:nn { #1 } { #2 }
2734     }
2735 }
2736 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2737 }
2738 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2739 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2740 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2741 {
2742     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2743     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2744     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2745     {
2746         \@@_error:nn { delimiter~after~opening } { #2 }
2747         \@@_rec_preamble:n
2748     }
2749     { \@@_rec_preamble:n #2 }
2750 }

```

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

```

2751 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2752 { \use:c { @@ _ \token_to_str:N ( : } }

```

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

```

2753 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2754 {
2755     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2756     \tl_if_in:nnTF { ) ] \} } { #2 }
2757     { \@@_make_preamble_v:nnn #1 #2 }
2758     {
2759         \str_if_eq:nnTF { \s_stop } { #2 }

```

```

2760     {
2761       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2762       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2763       {
2764         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2765         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2766         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2767         \@@_rec_preamble:n #2
2768       }
2769     }
2770     {
2771       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2772       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2773       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2774       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2775       \@@_rec_preamble:n #2
2776     }
2777   }
2778 }
2779 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2780 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2781 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2782 {
2783   \str_if_eq:nnTF { \s_stop } { #3 }
2784   {
2785     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2786     {
2787       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2788       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2789       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2790       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2791     }
2792     {
2793       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2794       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2795       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2796       \@@_error:nn { double~closing~delimiter } { #2 }
2797     }
2798   }
2799   {
2800     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2801     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2802     \@@_error:nn { double~closing~delimiter } { #2 }
2803     \@@_rec_preamble:n #3
2804   }
2805 }
2806 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2807 { \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 @{...}.

```

2808 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2809 {
2810   \str_if_eq:nnTF { #1 } { < }
2811   { \@@_rec_preamble_after_col_i:n }
2812   {
2813     \str_if_eq:nnTF { #1 } { @ }
2814     { \@@_rec_preamble_after_col_ii:n }
2815     {
2816       \str_if_eq:eeTF { \l_@@_vlines_clist } { all }

```

```

2817     {
2818         \tl_gput_right:Nn \g_@@_array_preamble_tl
2819         { ! { \skip_horizontal:N \arrayrulewidth } }
2820     }
2821     {
2822         \clist_if_in:NeT \l_@@_vlines_clist
2823         { \int_eval:n { \c@jCol + 1 } }
2824         {
2825             \tl_gput_right:Nn \g_@@_array_preamble_tl
2826             { ! { \skip_horizontal:N \arrayrulewidth } }
2827         }
2828     }
2829     \@@_rec_preamble:n { #1 }
2830 }
2831 }
2832 }
2833 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2834 {
2835     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2836     \@@_rec_preamble_after_col:n
2837 }

```

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.

```

2838 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2839 {
2840     \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
2841     {
2842         \tl_gput_right:Nn \g_@@_array_preamble_tl
2843         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2844     }
2845     {
2846         \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2847         {
2848             \tl_gput_right:Nn \g_@@_array_preamble_tl
2849             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2850         }
2851         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2852     }
2853     \@@_rec_preamble:n
2854 }
2855 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3
2856 {
2857     \tl_clear:N \l_tmpa_tl
2858     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2859     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2860 }

```

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.

```

2861 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2862 { \@@_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.

```

2863 \cs_new_protected:Npn \@@_X: #1 #2
2864 {
2865     \str_if_eq:nnTF { #2 } { [ ]
2866         { \@@_make_preamble_X:w [ ]
2867           { \@@_make_preamble_X:w [ ] #2 }
2868         }

```

```

2869 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2870 { \@@_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`.

```

2871 \keys_define:nn { nicematrix / X-column }
2872 {
2873   V .code:n =
2874     \IfPackageLoadedTF { varwidth }
2875     {
2876       \bool_set_true:N \l_@@_V_of_X_bool
2877       \bool_gset_true:N \g_@@_V_of_X_bool
2878     }
2879     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2880   unknown .code:n =
2881     \regex_if_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2882     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2883     { \@@_error_or_warning:n { invalid~weight } }
2884 }

```

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

```

2885 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2886 {

```

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).

```

2887   \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).

```

2888   \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`).

```

2889   \fp_set:Nn \l_tmpa_fp { 1.0 }

```

```

2890   \@@_keys_p_column:n { #1 }

```

The unknown keys have been stored by `\@@_keys_p_column:n` in `\l_tmpa_tl` and we use them right away in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitly provided by the final user.

```

2891   \bool_set_false:N \l_@@_V_of_X_bool
2892   \keys_set:no { nicematrix / X-column } \l_tmpa_tl

```

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

```

2893   \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).

```

2894   \bool_if:NTF \l_@@_X_columns_aux_bool
2895   {
2896     \@@_make_preamble_ii_iv:nnn

```

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

```

2897     { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2898     { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2899     { \@@_no_update_width: }
2900   }

```

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.

```

2901 {
2902   \tl_gput_right:Nn \g_@@_array_preamble_tl
2903   {
2904     > {
2905       \@@_cell_begin:
2906       \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`.

```

2907       \NotEmpty

```

The following code will nullify the box of the cell.

```

2908       \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2909       { \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`.

```

2910       \begin { minipage } { 5 cm } \arraybackslash
2911     }
2912     c
2913     < {
2914       \end { minipage }
2915       \@@_cell_end:
2916     }
2917   }
2918   \int_gincr:N \c@jCol
2919   \@@_rec_preamble_after_col:n
2920 }
2921 }

```

```

2922 \cs_new_protected:Npn \@@_no_update_width:
2923 {
2924   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2925   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2926 }

```

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

```

2927 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2928 {
2929   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2930   { \int_eval:n { \c@jCol + 1 } }
2931   \tl_gput_right:Ne \g_@@_array_preamble_tl
2932   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2933   \@@_rec_preamble:n
2934 }

```

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.

```

2935 \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).

```

2936 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2937 { \@@_fatal:n { Preamble-forgotten } }
2938 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2939 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2940 { @@ _ \token_to_str:N \hline : }
2941 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }

```

```

2942 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2943 { @@ _ \token_to_str:N \hline : }
2944 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2945 { @@ _ \token_to_str:N \hline : }
2946 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2947 { @@ _ \token_to_str:N \hline : }
2948 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2949 { @@ _ \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).

```

2950 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2951 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2952 \multispan { #1 }
2953 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2954 \begingroup
2955 \tbl_update_multicolumn_cell_data:n { #1 }

```

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

```

2956 \tl_gclear:N \g_@@_preamble_tl
2957 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2958 \def \@addamp
2959 {
2960 \legacy_if:nTF { @firstamp }
2961 { \legacy_if_set_false:n { @firstamp } }
2962 { \@preamerr 5 }
2963 }
2964 \exp_args:No \@mkpream \g_@@_preamble_tl
2965 \@addtopreamble \empty
2966 \endgroup
2967 \UseTaggingSocket { tbl / colspan } { #1 }

```

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

```

2968 \int_compare:nNnT { #1 } > { \c_one_int }
2969 {
2970 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2971 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2972 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2973 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2974 {
2975 {
2976 \int_if_zero:nTF { \c@jCol }
2977 { \int_eval:n { \c@iRow + 1 } }
2978 { \int_use:N \c@iRow }
2979 }
2980 { \int_eval:n { \c@jCol + 1 } }
2981 {
2982 \int_if_zero:nTF { \c@jCol }
2983 { \int_eval:n { \c@iRow + 1 } }
2984 { \int_use:N \c@iRow }

```

```

2985         }
2986         { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block.

```

2987         { }
2988     }
2989 }

```

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

```

2990 \RenewDocumentCommand { \cellcolor } { 0 { } m }
2991 {
2992     \tl_gput_right:Nn \g_@@_pre_code_before_tl
2993     {
2994         \@@_rectanglecolor [ ##1 ]
2995         { \exp_not:n { ##2 } }
2996         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2997         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2998     }
2999     \ignorespaces
3000 }

```

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

```

3001 \def \@sharp { #3 }
3002 \@arstrut
3003 \@preamble
3004 \null

```

We add some lines.

```

3005 \int_gadd:Nn \c@jCol { #1 - 1 }
3006 \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
3007 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
3008 \ignorespaces
3009 }

```

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`.

```

3010 \cs_new_protected:Npn \@@_make_m_preamble:n #1
3011 {
3012     \str_case:nnF { #1 }
3013     {
3014         c { \@@_make_m_preamble_i:n #1 }
3015         l { \@@_make_m_preamble_i:n #1 }
3016         r { \@@_make_m_preamble_i:n #1 }
3017         > { \@@_make_m_preamble_ii:nn #1 }
3018         ! { \@@_make_m_preamble_ii:nn #1 }
3019         @ { \@@_make_m_preamble_ii:nn #1 }
3020         | { \@@_make_m_preamble_iii:n #1 }
3021         p { \@@_make_m_preamble_iv:nnn t #1 }
3022         m { \@@_make_m_preamble_iv:nnn c #1 }
3023         b { \@@_make_m_preamble_iv:nnn b #1 }
3024         w { \@@_make_m_preamble_v:nnnn { } #1 }
3025         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
3026         \q_stop { }
3027     }
3028     {
3029         \cs_if_exist:cTF { NC @ find @ #1 }
3030         {
3031             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
3032             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
3033         }
3034         {

```

```

3035         \str_if_eq:nnTF { #1 } { S }
3036         { \@@_fatal:n { unknown~column~type~S~multicolumn } }
3037         { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
3038     }
3039 }
3040 }

```

For c, l and r

```

3041 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
3042 {
3043     \tl_gput_right:Nn \g_@@_preamble_tl
3044     {
3045         > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
3046         #1
3047         < \@@_cell_end:
3048     }

```

We test for the presence of a <.

```

3049     \@@_make_m_preamble_x:n
3050 }

```

For >, ! and @

```

3051 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
3052 {
3053     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
3054     \@@_make_m_preamble:n
3055 }

```

For l

```

3056 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
3057 {
3058     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
3059     \@@_make_m_preamble:n
3060 }

```

For p, m and b

```

3061 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
3062 {
3063     \tl_gput_right:Nn \g_@@_preamble_tl
3064     {
3065         > {
3066             \@@_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`.

```

3067         \setlength { \l_tmpa_dim } { #3 }
3068         \begin { minipage } [ #1 ] { \l_tmpa_dim }
3069         \mode_leave_vertical:
3070         \arraybackslash
3071         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
3072     }
3073     c
3074     < {
3075         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
3076         \end { minipage }
3077         \@@_cell_end:
3078     }
3079 }

```

We test for the presence of a <.

```

3080     \@@_make_m_preamble_x:n
3081 }

```



For w and W

```

3082 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
3083 {
3084   \tl_gput_right:Nn \g_@@_preamble_tl
3085   {
3086     > {
3087       \dim_set:Nn \l_@@_col_width_dim { #4 }
3088       \hbox_set:Nw \l_@@_cell_box
3089       \@@_cell_begin:
3090       \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3091     }
3092     c
3093     < {
3094       \@@_cell_end:
3095       \hbox_set_end:
3096       \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3097       #1
3098       \@@_adjust_size_box:
3099       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3100     }
3101   }

```

We test for the presence of a <.

```

3102   \@@_make_m_preamble_x:n
3103 }

```

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

```

3104 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3105 {
3106   \str_if_eq:nnTF { #1 } { < }
3107   { \@@_make_m_preamble_ix:n }
3108   { \@@_make_m_preamble:n { #1 } }
3109 }
3110 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3111 {
3112   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3113   \@@_make_m_preamble_x:n
3114 }

```

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).

```

3115 \cs_new_protected:Npn \@@_put_box_in_flow:
3116 {
3117   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3118   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3119   \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
3120   { \box_use_drop:N \l_tmpa_box }
3121   { \@@_put_box_in_flow_i: }
3122 }

```

The command \@@\_put\_box\_in\_flow\_i: is used when the value of \l\_@@\_baseline\_tl is different of c (the initial value).

```

3123 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3124 {
3125   \pgfpicture
3126   \@@_qpoint:n { row - 1 }
3127   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3128   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3129   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3130   \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).

```

3131 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3132 {
3133   \int_set:Nn \l_tmpa_int
3134   { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
3135   \bool_lazy_or:nnT
3136   { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
3137   { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
3138   {
3139     \@@_error:n { bad-value-for-baseline-line }
3140     \int_set_eq:NN \l_tmpa_int \c_one_int
3141   }
3142   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3143 }
3144 {
3145   \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
3146   { \int_set_eq:NN \l_tmpa_int \c_one_int }
3147   {
3148     \str_if_eq:onTF \l_@@_baseline_tl { b }
3149     { \int_set_eq:NN \l_tmpa_int \c@iRow }
3150     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3151   }
3152   \bool_lazy_or:nnT
3153   { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3154   { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3155   {
3156     \@@_error:n { bad-value-for-baseline }
3157     \int_set_eq:NN \l_tmpa_int \c_one_int
3158   }
3159   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3160 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3161 }
3162 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3163 \endpgfpicture
3164 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3165 \box_use_drop:N \l_tmpa_box
3166 }

```

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).

```

3167 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3168 {

```

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.

```

3169 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3170 {
3171   \int_compare:nNnT { \c@jCol } > { \c_one_int }
3172   {
3173     \box_set_wd:Nn \l_@@_the_array_box
3174     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3175   }
3176 }

```

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

```

3177 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }

```

```

3178 \bool_if:NT \l_@@_caption_above_bool
3179 {
3180   \tl_if_empty:NF \l_@@_caption_tl
3181   {
3182     \bool_set_false:N \g_@@_caption_finished_bool
3183     \int_gzero:N \c@tabularnote
3184     \@@_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).

```

3185     \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3186     {
3187       \tl_gput_right:Ne \g_@@_aux_tl
3188       {
3189         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3190         { \int_use:N \g_@@_notes_caption_int }
3191       }
3192       \int_gzero:N \g_@@_notes_caption_int
3193     }
3194   }
3195 }

```

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

```

3196 \hbox
3197 {
3198   \box_use_drop:N \l_@@_the_array_box

```

We have to draw the blocks right away because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are `medium` nodes to create for the blocks.

```

3199   \@@_create_extra_nodes:
3200   \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3201 }

```

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).

```

3202 \bool_lazy_any:nT
3203 {
3204   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3205   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3206   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3207 }
3208 \@@_insert_tabularnotes:
3209 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3210 \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3211 \end { minipage }
3212 }

```

```

3213 \cs_new_protected:Npn \@@_insert_caption:
3214 {
3215   \tl_if_empty:NF \l_@@_caption_tl
3216   {
3217     \cs_if_exist:NTF \@capttype
3218     { \@@_insert_caption_i: }
3219     { \@@_error:n { caption~outside~float } }
3220   }
3221 }

```

```

3222 \cs_new_protected:Npn \@@_insert_caption_i:
3223 {
3224   \group_begin:

```

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

```

3225   \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.

```

3226   \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
3227   \tl_if_empty:NTF \l_@@_short_caption_tl
3228     { \caption }
3229     { \caption [ \l_@@_short_caption_tl ] }
3230     { \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.

```

3231   \bool_if:NF \g_@@_caption_finished_bool
3232   {
3233     \bool_gset_true:N \g_@@_caption_finished_bool
3234     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3235     \int_gzero:N \c@tabularnote
3236   }
3237   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3238   \group_end:
3239 }

```

```

3240 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3241 {
3242   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3243   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3244 }

```

```

3245 \cs_new_protected:Npn \@@_insert_tabularnotes:
3246 {
3247   \seq_concat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3248   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3249   \skip_vertical:N 0.65ex

```

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

```

3250   \group_begin:
3251   \l_@@_notes_code_before_tl
3252   \tl_if_empty:NF \g_@@_tabularnote_tl
3253   {
3254     \g_@@_tabularnote_tl \par
3255     \tl_gclear:N \g_@@_tabularnote_tl
3256   }

```

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.

```

3257   \int_compare:nNtT { \c@tabularnote } > { \c_zero_int }
3258   {
3259     \bool_if:NTF \l_@@_notes_para_bool
3260     {
3261       \begin { tabularnotes* }
3262         \seq_map_inline:Nn \g_@@_notes_seq
3263           { \@@_one_tabularnote:nn ##1 }
3264         \strut
3265       \end { tabularnotes* }

```

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

```

3266         \par
3267     }
3268     {
3269         \tabularnotes
3270         \seq_map_inline:Nn \g_@@_notes_seq
3271         { \@@_one_tabularnote:nn ##1 }
3272         \strut
3273         \endtabularnotes
3274     }
3275 }
3276 \unskip
3277 \group_end:
3278 \bool_if:NT \l_@@_notes_bottomrule_bool
3279 {
3280     \IfPackageLoadedTF { booktabs }
3281     {

```

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

```

3282         \skip_vertical:N \aboverulesep

```

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

```

3283         { \CT@arc@ \hrule height \heavyrulewidth }
3284     }
3285     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3286 }
3287 \l_@@_notes_code_after_tl
3288 \seq_gclear:N \g_@@_notes_seq
3289 \seq_gclear:N \g_@@_notes_in_caption_seq
3290 \int_gzero:N \c@tabularnote
3291 }

```

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 curryfication.

```

3292 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3293 {
3294     \tl_if_novalue:nTF { #1 }
3295     { \item }
3296     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3297 }

```

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`.

```

3298 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3299 {
3300     \pgfpicture
3301     \@@_qpoint:n { row - 1 }
3302     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3303     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3304     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3305     \endpgfpicture
3306     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3307     \int_if_zero:nT { \l_@@_first_row_int }
3308     {
3309         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3310         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3311     }
3312     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3313 }

```

Now, the general case.

```
3314 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3315 {
```

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

```
3316 \str_if_eq:eeT { \l_@@_baseline_tl } { t }
3317 { \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  $\l\_tmpa\_int$ .

```
3318 \pgfpicture
3319 \@@_qpoint:n { row - 1 }
3320 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3321 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3322 {
3323   \int_set:Nn \l_tmpa_int
3324   {
3325     \str_range:Nnn
3326     \l_@@_baseline_tl
3327     { 6 }
3328     { \tl_count:o \l_@@_baseline_tl }
3329   }
3330   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3331 }
3332 {
3333   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3334   \bool_lazy_or:nnT
3335   { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3336   { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3337   {
3338     \@@_error:n { bad~value~for~baseline }
3339     \int_set:Nn \l_tmpa_int 1
3340   }
3341   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3342 }
3343 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3344 \endpgfpicture
3345 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3346 \int_if_zero:nT { \l_@@_first_row_int }
3347 {
3348   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3349   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3350 }
3351 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3352 }
```

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.

```
3353 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3354 {
```

We will compute the real width of both delimiters used.

```
3355 \dim_zero_new:N \l_@@_real_left_delim_dim
3356 \dim_zero_new:N \l_@@_real_right_delim_dim
3357 \hbox_set:Nn \l_tmpb_box
3358 {
3359   \m@th
3360   $ % $
3361   \left #1
3362   \vcenter
3363   {
3364     \vbox_to_ht:nn
3365     { \box_ht_plus_dp:N \l_tmpa_box }
```

```

3366         { }
3367     }
3368     \right .
3369     $ % $
3370 }
3371 \dim_set:Nn \l_@@_real_left_delim_dim
3372 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3373 \hbox_set:Nn \l_tmpb_box
3374 {
3375     \m@th
3376     $ % $
3377     \left .
3378     \vbox_to_ht:nn
3379     { \box_ht_plus_dp:N \l_tmpa_box }
3380     { }
3381     \right #2
3382     $ % $
3383 }
3384 \dim_set:Nn \l_@@_real_right_delim_dim
3385 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3386     \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3387     \@@_put_box_in_flow:
3388     \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3389 }

```

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).

```

3390 \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).

```

3391 {
3392     \peek_remove_spaces:n
3393     {
3394         \peek_meaning:NTF \end
3395         { \@@_analyze_end:Nn }
3396         {
3397             \@@_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`).

```

3398         \@@_array:o \g_@@_array_preamble_tl
3399     }
3400 }
3401 {
3402     {
3403         \@@_create_col_nodes:
3404         \endarray
3405     }

```

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

```

3406 \NewDocumentEnvironment { @@-light-syntax } { b }
3407 {

```

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.

```

3408 \tl_if_empty:nT { #1 }
3409 { \@@_fatal:n { empty~environment } }
3410 \tl_if_in:nnT { #1 } { & }
3411 { \@@_fatal:n { ampersand~in~light-syntax } }
3412 \tl_if_in:nnT { #1 } { \ }
3413 { \@@_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`.

```

3414 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3415 }

```

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.

```

3416 {
3417 \@@_create_col_nodes:
3418 \endarray
3419 }
3420 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3421 {
3422 \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).

```

3423 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3424 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3425 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3426 { \seq_set_split:Nee }
3427 { \seq_set_split:Nnon }
3428 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3429 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3430 \tl_if_empty:NF \l_tmpa_tl
3431 { \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.

```

3432 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3433 { \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`).

```

3434 \tl_build_begin:N \l_@@_new_body_tl
3435 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3436 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3437 \@@_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).

```

3438 \seq_map_inline:Nn \l_@@_rows_seq
3439 {
3440   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3441   \@@_line_with_light_syntax:n { #1 }
3442 }
3443 \tl_build_end:N \l_@@_new_body_tl
3444 \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3445 {
3446   \int_set:Nn \l_@@_last_col_int
3447   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3448 }

```

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.

```

3449 \@@_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`).

```

3450 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3451 }
3452 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3453 {
3454   \seq_clear_new:N \l_@@_cells_seq
3455   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3456   \int_set:Nn \l_@@_nb_cols_int
3457   {
3458     \int_max:nn
3459     { \l_@@_nb_cols_int }
3460     { \seq_count:N \l_@@_cells_seq }
3461   }
3462   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3463   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3464   \seq_map_inline:Nn \l_@@_cells_seq
3465   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3466 }
3467 \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`.

```

3468 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3469 {
3470   \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
3471   { \@@_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.

```

3472   \end { #2 }
3473 }

```

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`).

```

3474 \cs_new:Npn \@@_create_col_nodes:
3475 {
3476   \crrc
3477   \int_if_zero:nT { \l_@@_first_col_int }
3478   {
3479     \omit

```

```

3480 \hbox_overlap_left:n
3481 {
3482   \bool_if:NT \l_@@_code_before_bool
3483   { \pgfsys@markposition { \@@_env: - col - 0 } }
3484   \pgfpicture
3485   \pgfrememberpicturepositiononpagetrue
3486   \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3487   \str_if_empty:NF \l_@@_name_str
3488   { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3489   \endpgfpicture
3490   \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3491 }
3492 &
3493 }
3494 \omit

```

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

```

3495 \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`).

```

3496 \int_if_zero:nTF { \l_@@_first_col_int }
3497 {
3498   \@@_mark_position:n { 1 }
3499   \pgfpicture
3500   \pgfrememberpicturepositiononpagetrue
3501   \pgfcoordinate { \@@_env: - col - 1 }
3502   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3503   \str_if_empty:NF \l_@@_name_str
3504   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3505   \endpgfpicture
3506 }
3507 {
3508   \bool_if:NT \l_@@_code_before_bool
3509   {
3510     \hbox
3511     {
3512       \skip_horizontal:n { 0.5 \arrayrulewidth }
3513       \pgfsys@markposition { \@@_env: - col - 1 }
3514       \skip_horizontal:n { -0.5 \arrayrulewidth }
3515     }
3516   }
3517   \pgfpicture
3518   \pgfrememberpicturepositiononpagetrue
3519   \pgfcoordinate { \@@_env: - col - 1 }
3520   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3521   \@@_node_alias:n { 1 }
3522   \endpgfpicture
3523 }

```

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.

```

3524 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3525 \bool_if:NF \l_@@_auto_columns_width_bool
3526 { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3527 {
3528   \bool_lazy_and:nnTF
3529   { \l_@@_auto_columns_width_bool }
3530   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3531   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }

```

```

3532         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3533     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3534 }
3535 \skip_horizontal:N \g_tmpa_skip
3536 \hbox
3537 {
3538     \@@_mark_position:n { 2 }
3539     \pgfpicture
3540     \pgfrememberpicturepositiononpagetrue
3541     \pgfcoordinate { \@@_env: - col - 2 }
3542     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3543     \@@_node_alias:n { 2 }
3544     \endpgfpicture
3545 }

```

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.

```

3546 \int_gset_eq:NN \g_tmpa_int \c_one_int
3547 \bool_if:NTF \g_@@_last_col_found_bool
3548 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3549 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3550 {
3551     &
3552     \omit
3553     \int_gincr:N \g_tmpa_int

```

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

```

3554     \skip_horizontal:N \g_tmpa_skip
3555     \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

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

```

3556 \pgfpicture
3557 \pgfrememberpicturepositiononpagetrue
3558 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3559 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3560 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3561 \endpgfpicture
3562 }

```

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).

```

3563 \bool_lazy_or:nnF
3564 { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3565 { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3566 {
3567     &
3568     \omit
3569     \skip_horizontal:N \g_tmpa_skip
3570     \int_gincr:N \g_tmpa_int
3571     \bool_lazy_any:nF
3572     {
3573         \g_@@_delims_bool
3574         \l_@@_tabular_bool
3575         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3576         \l_@@_exterior_arraycolsep_bool
3577         \l_@@_bar_at_end_of_pream_bool
3578     }
3579     { \skip_horizontal:n { - \col@sep } }
3580     \bool_if:NT \l_@@_code_before_bool
3581     {
3582         \hbox
3583         {
3584             \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.

```

3585         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3586         { \skip_horizontal:n { - \arraycolsep } }
3587     \pgfsys@markposition
3588     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3589     \skip_horizontal:n { 0.5 \arrayrulewidth }
3590     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3591     { \skip_horizontal:N \arraycolsep }
3592 }
3593 }
3594 \pgfpicture
3595 \pgfrememberpicturepositiononpagetrue
3596 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3597 {
3598     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3599     {
3600         \pgfpoint
3601         { - 0.5 \arrayrulewidth - \arraycolsep }
3602         \c_zero_dim
3603     }
3604     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3605 }
3606 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3607 \endpgfpicture
3608 }

3609 \bool_if:NT \g_@@_last_col_found_bool
3610 {
3611     \hbox_overlap_right:n
3612     {
3613         \skip_horizontal:N \g_@@_width_last_col_dim
3614         \skip_horizontal:N \col@sep
3615         \bool_if:NT \l_@@_code_before_bool
3616         {
3617             \pgfsys@markposition
3618             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3619         }
3620         \pgfpicture
3621         \pgfrememberpicturepositiononpagetrue
3622         \pgfcoordinate
3623         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3624         \pgfpointorigin
3625         \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3626         \endpgfpicture
3627     }
3628 }
3629 }

3630 \cs_new_protected:Npn \@@_mark_position:n #1
3631 {
3632     \bool_if:NT \l_@@_code_before_bool
3633     {
3634         \hbox
3635         {
3636             \skip_horizontal:n { -0.5 \arrayrulewidth }
3637             \pgfsys@markposition { \@@_env: - col - #1 }
3638             \skip_horizontal:n { 0.5 \arrayrulewidth }
3639         }
3640     }
3641 }

```

```

3642 \cs_new_protected:Npn \@@_node_alias:n #1
3643 {
3644   \str_if_empty:NF \l_@@_name_str
3645   { \pgfnodelalias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3646 }

```

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

```

3647 \tl_const:Nn \c_@@_preamble_first_col_tl
3648 {
3649   >
3650   {

```

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

```

3651     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3652     \bool_gset_true:N \g_@@_after_col_zero_bool
3653     \@@_begin_of_row:
3654     \hbox_set:Nw \l_@@_cell_box
3655     \@@_math_toggle:
3656     \@@_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”.

```

3657     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3658     {
3659       \bool_lazy_or:nnT
3660       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3661       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3662       {
3663         \l_@@_code_for_first_col_tl
3664         \xglobal \colorlet { nicematrix-first-col } { . }
3665       }
3666     }
3667 }

```

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`.

```

3668   l
3669   <
3670   {
3671     \@@_math_toggle:
3672     \hbox_set_end:
3673     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3674     \@@_adjust_size_box:
3675     \@@_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.

```

3676     \dim_gset:Nn \g_@@_width_first_col_dim
3677     { \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.

```

3678     \hbox_overlap_left:n
3679     {
3680       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3681       { \@@_node_cell: }
3682       { \box_use_drop:N \l_@@_cell_box }
3683       \skip_horizontal:N \l_@@_left_delim_dim
3684       \skip_horizontal:N \l_@@_left_margin_dim
3685       \skip_horizontal:N \l_@@_extra_left_margin_dim
3686     }
3687     \bool_gset_false:N \g_@@_empty_cell_bool
3688     \skip_horizontal:n { -2 \col@sep }
3689   }
3690 }

```

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

```

3691 \tl_const:Nn \c_@@_preamble_last_col_tl
3692 {
3693   >
3694   {
3695     \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).

```

3696     \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.

```

3697     \bool_gset_true:N \g_@@_last_col_found_bool
3698     \int_gincr:N \c@jCol
3699     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3700     \hbox_set:Nw \l_@@_cell_box
3701     \@@_math_toggle:
3702     \@@_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”.

```

3703     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3704     {
3705       \bool_lazy_or:nnT
3706       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3707       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3708       {
3709         \l_@@_code_for_last_col_tl
3710         \xglobal \colorlet { nicematrix-last-col } { . }
3711       }
3712     }
3713   }
3714   l
3715   <
3716   {
3717     \@@_math_toggle:
3718     \hbox_set_end:
3719     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3720     \@@_adjust_size_box:
3721     \@@_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.

```

3722     \dim_gset:Nn \g_@@_width_last_col_dim
3723     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3724     \skip_horizontal:n { -2 \col@sep }

```

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

```

3725     \hbox_overlap_right:n
3726     {
3727       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3728       {
3729         \skip_horizontal:N \l_@@_right_delim_dim
3730         \skip_horizontal:N \l_@@_right_margin_dim
3731         \skip_horizontal:N \l_@@_extra_right_margin_dim
3732         \@@_node_cell:
3733       }
3734     }
3735     \bool_gset_false:N \g_@@_empty_cell_bool
3736   }
3737 }

```

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

```

3738 \NewDocumentEnvironment { NiceArray } { }

```

```

3739 {
3740   \bool_gset_false:N \g_@@_delims_bool
3741   \str_if_empty:NT \g_@@_name_env_str
3742   { \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).

```

3743   \NiceArrayWithDelims . .
3744 }
3745 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3746 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3747 {
3748   \NewDocumentEnvironment { #1 NiceArray } { }
3749   {
3750     \bool_gset_true:N \g_@@_delims_bool
3751     \str_if_empty:NT \g_@@_name_env_str
3752     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3753     \@@_test_if_math_mode:
3754     \NiceArrayWithDelims #2 #3
3755   }
3756   { \endNiceArrayWithDelims }
3757 }
3758 \@@_def_env:NNN p ( )
3759 \@@_def_env:NNN b [ ]
3760 \@@_def_env:NNN B \{ \}
3761 \@@_def_env:NNN v \vert \vert
3762 \@@_def_env:NNN V \Vert \Vert

```

## 13 The environment {NiceMatrix} and its variants

```

3763 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3764 {
3765   \bool_set_false:N \l_@@_preamble_bool
3766   \tl_clear:N \l_tmpa_tl
3767   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3768   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3769   \tl_put_right:Nn \l_tmpa_tl
3770   {
3771     *
3772     {
3773       \int_case:nnF \l_@@_last_col_int
3774       {
3775         { -2 } { \c@MaxMatrixCols }
3776         { -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).

```

3777       }
3778       { \int_eval:n { \l_@@_last_col_int - 1 } }
3779     }
3780     { #2 }
3781   }
3782   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3783   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3784 }
3785 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3786 \clist_map_inline:nn { p , b , B , v , V }

```

```

3787 {
3788   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3789   {
3790     \bool_gset_true:N \g_@@_delims_bool
3791     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3792     \int_if_zero:nT { \l_@@_last_col_int }
3793     {
3794       \bool_set_true:N \l_@@_last_col_without_value_bool
3795       \int_set:Nn \l_@@_last_col_int { -1 }
3796     }
3797     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3798     \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3799   }
3800   { \use:c { end #1 NiceArray } }
3801 }

```

We define also an environment {NiceMatrix}

```

3802 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3803 {
3804   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3805   \int_if_zero:nT { \l_@@_last_col_int }
3806   {
3807     \bool_set_true:N \l_@@_last_col_without_value_bool
3808     \int_set:Nn \l_@@_last_col_int { -1 }
3809   }
3810   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3811   \bool_lazy_or:nnT
3812     { \clist_if_empty_p:N \l_@@_vlines_clist }
3813     { \l_@@_except_borders_bool }
3814     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3815   \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3816 }
3817 { \endNiceArray }

```

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

```

3818 \cs_new_protected:Npn \@@_NotEmpty:
3819 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3820 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3821 {

```

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.

```

3822   \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
3823     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3824   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3825   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3826   \tl_if_empty:NF \l_@@_short_caption_tl
3827   {
3828     \tl_if_empty:NT \l_@@_caption_tl
3829     {
3830       \@@_error_or_warning:n { short-caption-without~caption }
3831       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3832     }
3833   }
3834   \tl_if_empty:NF \l_@@_label_tl
3835   {
3836     \tl_if_empty:NT \l_@@_caption_tl
3837     { \@@_error_or_warning:n { label~without~caption } }

```



```

3838     }
3839     \NewDocumentEnvironment { TabularNote } { b }
3840     {
3841         \bool_if:NTF \l_@@_in_code_after_bool
3842         { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3843         {
3844             \tl_if_empty:NF \g_@@_tabularnote_tl
3845             { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3846             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3847         }
3848     }
3849     { }
3850     \@@_settings_for_tabular:
3851     \NiceArray { #2 }
3852 }
3853 { \endNiceArray }
3854 \cs_new_protected:Npn \@@_settings_for_tabular:
3855 {
3856     \bool_set_true:N \l_@@_tabular_bool
3857     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3858     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3859     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3860 }

3861 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3862 {
3863     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3864     \dim_set:Nn \l_@@_width_dim { #1 }
3865     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3866     \@@_settings_for_tabular:
3867     \NiceArray { #3 }
3868 }
3869 {
3870     \endNiceArray
3871     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3872     { \@@_error:n { NiceTabularX~without~X } }
3873 }

3874 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3875 {
3876     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3877     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3878     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3879     \@@_settings_for_tabular:
3880     \NiceArray { #3 }
3881 }
3882 { \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`).

```

3883 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3884 {
3885     \bool_lazy_all:nT
3886     {
3887         { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3888         { \l_@@_hvlines_bool }

```

```

3889     { ! \g_@@_delims_bool }
3890     { ! \l_@@_except_borders_bool }
3891   }
3892   {
3893     \bool_set_true:N \l_@@_except_borders_bool
3894     \clist_if_empty:NF \l_@@_corners_clist
3895     { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3896     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3897     {
3898       \@@_stroke_block:nnn
3899       {
3900         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3901         draw = \l_@@_rules_color_tl
3902       }
3903       { 1-1 }
3904       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3905     }
3906   }
3907 }

3908 \cs_new_protected:Npn \@@_after_array:
3909 {

```

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`.

```

3910   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3911   \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.

```

3912   \bool_if:NT \g_@@_last_col_found_bool
3913   { \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`.

```

3914   \bool_if:NT \l_@@_last_col_without_value_bool
3915   { \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.

```

3916   \bool_if:NT \l_@@_last_row_without_value_bool
3917   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3918   \tl_gput_right:Ne \g_@@_aux_tl
3919   {
3920     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3921     {
3922       \int_use:N \l_@@_first_row_int ,
3923       \int_use:N \c@iRow ,
3924       \int_use:N \g_@@_row_total_int ,
3925       \int_use:N \l_@@_first_col_int ,
3926       \int_use:N \c@jCol ,
3927       \int_use:N \g_@@_col_total_int
3928     }
3929   }

```

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`).

```

3930 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3931 {
3932   \tl_gput_right:Ne \g_@@_aux_tl
3933   {
3934     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3935     { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
3936   }
3937 }
3938 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3939 {
3940   \tl_gput_right:Ne \g_@@_aux_tl
3941   {
3942     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3943     { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
3944     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3945     { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
3946   }
3947 }

```

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

```

3948 \@@_create_diag_nodes:

```

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

```

3949 \pgfpicture
3950 \@@_create_aliases_last:
3951 \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3952 \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.

```

3953 \bool_if:NT \l_@@_parallelize_diags_bool
3954 {
3955   \int_gzero:N \g_@@_ddots_int
3956   \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.

```

3957   \dim_gzero:N \g_@@_delta_x_one_dim
3958   \dim_gzero:N \g_@@_delta_y_one_dim
3959   \dim_gzero:N \g_@@_delta_x_two_dim
3960   \dim_gzero:N \g_@@_delta_y_two_dim
3961 }
3962 \bool_set_false:N \l_@@_initial_open_bool
3963 \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.

```

3964 \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

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

```

3965 \@@_draw_dotted_lines:

```

---

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

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

```

3966   \clist_if_empty:NF \l_@@_corners_clist
3967   {
3968     \bool_if:NTF \l_@@_no_cell_nodes_bool
3969     { \@@_error:n { corners~with~no~cell~nodes } }
3970     { \@@_compute_corners: }
3971   }

```

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

```

3972   \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
3973   \g_@@_pos_of_blocks_seq
3974   \g_@@_future_pos_of_blocks_seq
3975   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

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

```

3976   \@@_adjust_pos_of_blocks_seq:
3977   \@@_deal_with_rounded_corners:
3978   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3979   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

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

```

3980   \IfPackageLoadedT { tikz }
3981   {
3982     \tikzset
3983     {
3984       every~picture / .style =
3985       {
3986         overlay ,
3987         remember~picture ,
3988         name~prefix = \@@_env: -
3989       }
3990     }
3991   }
3992   \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
3993   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3994   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3995   \cs_set_eq:NN \OverBrace \@@_OverBrace
3996   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3997   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3998   \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.

```

3999   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
4000   \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.

```

4001   \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).

```

4002   \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`).

```
4003 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
4004 { \@@_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:`.

```
4005 \bool_set_true:N \l_@@_in_code_after_bool
4006 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
4007 \scan_stop:
4008 \tl_gclear:N \g_nicematrix_code_after_tl
4009 \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.

```
4010 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
4011 \tl_if_empty:NF \g_@@_pre_code_before_tl
4012 {
4013   \tl_gput_right:Ne \g_@@_aux_tl
4014   {
4015     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
4016     { \exp_not:o \g_@@_pre_code_before_tl }
4017   }
4018   \tl_gclear:N \g_@@_pre_code_before_tl
4019 }
4020 \tl_if_empty:NF \g_nicematrix_code_before_tl
4021 {
4022   \tl_gput_right:Ne \g_@@_aux_tl
4023   {
4024     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
4025     { \exp_not:o \g_nicematrix_code_before_tl }
4026   }
4027   \tl_gclear:N \g_nicematrix_code_before_tl
4028 }

4029 \str_gclear:N \g_@@_name_env_str
4030 \@@_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`.

```
4031 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4032 }

4033 \cs_new_protected:Npn \@@_tuning_key_small_for_dots:
4034 {
4035   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
4036   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

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

4037 \dim_set:Nn \l_@@_xdots_shorten_start_dim
4038 { 0.6 \l_@@_xdots_shorten_start_dim }
4039 \dim_set:Nn \l_@@_xdots_shorten_end_dim
4040 { 0.6 \l_@@_xdots_shorten_end_dim }
4041 }
```

---

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

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`.

```

4042 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4043 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

4044 \cs_new_protected:Npn \@@_create_alias_nodes:
4045 {
4046   \int_step_inline:nn { \c@iRow }
4047   {
4048     \pgfnodealias
4049     { \l_@@_name_str - ##1 - last }
4050     { \@@_env: - ##1 - \int_use:N \c@jCol }
4051   }
4052   \int_step_inline:nn { \c@jCol }
4053   {
4054     \pgfnodealias
4055     { \l_@@_name_str - last - ##1 }
4056     { \@@_env: - \int_use:N \c@iRow - ##1 }
4057   }
4058   \pgfnodealias
4059   { \l_@@_name_str - last - last }
4060   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
4061 }

```

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).

```

4062 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4063 {
4064   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4065   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4066 }

```

The following command must *not* be protected.

```

4067 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4068 {
4069   { #1 }
4070   { #2 }
4071   {
4072     \int_compare:nNnTF { #3 } > { 98 }
4073     { \int_use:N \c@iRow }
4074     { #3 }
4075   }
4076   {
4077     \int_compare:nNnTF { #4 } > { 98 }
4078     { \int_use:N \c@jCol }
4079     { #4 }
4080   }
4081   { #5 }
4082 }

```

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).

```

4083 \hook_gput_code:nnn { begindocument } { . }
4084 {

```

```

4085 \cs_new_protected:Npe \@@_draw_dotted_lines:
4086 {
4087   \c_@@_pgfortikzpicture_tl
4088   \@@_draw_dotted_lines_i:
4089   \c_@@_endpgfortikzpicture_tl
4090 }
4091 }

```

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

```

4092 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4093 {
4094   \pgfrememberpicturepositiononpagetrue
4095   \pgf@relevantforpicturesizefalse
4096   \g_@@_HVdotsfor_lines_tl
4097   \g_@@_Vdots_lines_tl
4098   \g_@@_Ddots_lines_tl
4099   \g_@@_Iddots_lines_tl
4100   \g_@@_Cdots_lines_tl
4101   \g_@@_Ldots_lines_tl
4102 }

4103 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4104 {
4105   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4106   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4107 }

```

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

```

4108 \pgfdeclareshape { @@_diag_node }
4109 {
4110   \savedanchor { \five }
4111   {
4112     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4113     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4114   }
4115   \anchor { 5 } { \five }
4116   \anchor { center } { \pgfpointorigin }
4117   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4118   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4119   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4120   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4121   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4122   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4123   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4124   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4125   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4126   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4127 }

```

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).

```

4128 \cs_new_protected:Npn \@@_create_diag_nodes:
4129 {
4130   \pgfpicture
4131   \pgfrememberpicturepositiononpagetrue
4132   \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
4133   {
4134     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4135     \dim_set_eq:NN \l_tmpa_dim \pgf@x

```

```

4136 \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4137 \dim_set_eq:NN \l_tmpb_dim \pgf@y
4138 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4139 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4140 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4141 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4142 \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.

```

4143 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4144 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4145 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4146 \str_if_empty:NF \l_@@_name_str
4147 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4148 }

```

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

```

4149 \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
4150 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4151 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4152 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4153 \pgfcoordinate
4154 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4155 \pgfnodealias
4156 { \@@_env: - last }
4157 { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
4158 \str_if_empty:NF \l_@@_name_str
4159 {
4160 \pgfnodealias
4161 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4162 { \@@_env: - \int_use:N \l_tmpa_int }
4163 \pgfnodealias
4164 { \l_@@_name_str - last }
4165 { \@@_env: - last }
4166 }
4167 \endpgfpicture
4168 }

```

## 16 We draw the dotted lines

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

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

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

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

This command computes:



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

```
4169 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4170 {
```

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

```
4171 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4172 \int_set:Nn \l_@@_initial_i_int { #1 }
4173 \int_set:Nn \l_@@_initial_j_int { #2 }
4174 \int_set:Nn \l_@@_final_i_int { #1 }
4175 \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.

```
4176 \bool_set_false:N \l_@@_stop_loop_bool
4177 \bool_do_until:Nn \l_@@_stop_loop_bool
4178 {
4179   \int_add:Nn \l_@@_final_i_int { #3 }
4180   \int_add:Nn \l_@@_final_j_int { #4 }
4181   \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.

```
4182 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4183   \if_int_compare:w #3 = \c_one_int
4184     \bool_set_true:N \l_@@_final_open_bool
4185   \else:
4186     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4187       \bool_set_true:N \l_@@_final_open_bool
4188     \fi:
4189   \fi:
4190 \else:
4191   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4192     \if_int_compare:w #4 = -1
4193       \bool_set_true:N \l_@@_final_open_bool
4194     \fi:
4195   \else:
4196     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4197       \if_int_compare:w #4 = \c_one_int
4198         \bool_set_true:N \l_@@_final_open_bool
4199       \fi:
4200     \fi:
4201   \fi:
4202 \fi:
4203 \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.

```
4204 {
```

We do a step backwards.

```
4205 \int_sub:Nn \l_@@_final_i_int { #3 }
4206 \int_sub:Nn \l_@@_final_j_int { #4 }
4207 \bool_set_true:N \l_@@_stop_loop_bool
4208 }
```

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`.

```

4209     {
4210         \cs_if_exist:cTF
4211         {
4212             @@ _ dotted _
4213             \int_use:N \l_@@_final_i_int -
4214             \int_use:N \l_@@_final_j_int
4215         }
4216         {
4217             \int_sub:Nn \l_@@_final_i_int { #3 }
4218             \int_sub:Nn \l_@@_final_j_int { #4 }
4219             \bool_set_true:N \l_@@_final_open_bool
4220             \bool_set_true:N \l_@@_stop_loop_bool
4221         }
4222     }
4223     \cs_if_exist:cTF
4224     {
4225         pgf @ sh @ ns @ \@@_env:
4226         - \int_use:N \l_@@_final_i_int
4227         - \int_use:N \l_@@_final_j_int
4228     }
4229     { \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.

```

4230     {
4231         \cs_set_nopar:cpn
4232         {
4233             @@ _ dotted _
4234             \int_use:N \l_@@_final_i_int -
4235             \int_use:N \l_@@_final_j_int
4236         }
4237         { }
4238     }
4239 }
4240 }
4241 }

```

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

```

4242     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4243     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4244     \bool_do_until:Nn \l_@@_stop_loop_bool
4245     {
4246         \int_sub:Nn \l_@@_initial_i_int { #3 }
4247         \int_sub:Nn \l_@@_initial_j_int { #4 }
4248         \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.

```

4249     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4250     \if_int_compare:w #3 = \c_one_int
4251         \bool_set_true:N \l_@@_initial_open_bool
4252     \else:

```

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

```

4253     \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4254     \bool_set_true:N \l_@@_initial_open_bool
4255     \fi:
4256   \fi:
4257 \else:
4258   \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4259   \if_int_compare:w #4 = \c_one_int
4260   \bool_set_true:N \l_@@_initial_open_bool
4261   \fi:
4262 \else:
4263   \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4264   \if_int_compare:w #4 = -1
4265   \bool_set_true:N \l_@@_initial_open_bool
4266   \fi:
4267 \fi:
4268 \fi:
4269 \fi:
4270 \bool_if:NTF \l_@@_initial_open_bool
4271 {
4272   \int_add:Nn \l_@@_initial_i_int { #3 }
4273   \int_add:Nn \l_@@_initial_j_int { #4 }
4274   \bool_set_true:N \l_@@_stop_loop_bool
4275 }
4276 {
4277   \cs_if_exist:cTF
4278   {
4279     @@ _ dotted _
4280     \int_use:N \l_@@_initial_i_int -
4281     \int_use:N \l_@@_initial_j_int
4282   }
4283   {
4284     \int_add:Nn \l_@@_initial_i_int { #3 }
4285     \int_add:Nn \l_@@_initial_j_int { #4 }
4286     \bool_set_true:N \l_@@_initial_open_bool
4287     \bool_set_true:N \l_@@_stop_loop_bool
4288   }
4289   {
4290     \cs_if_exist:cTF
4291     {
4292       pgf @ sh @ ns @ \@@_env:
4293       - \int_use:N \l_@@_initial_i_int
4294       - \int_use:N \l_@@_initial_j_int
4295     }
4296     { \bool_set_true:N \l_@@_stop_loop_bool }
4297     {
4298       \cs_set_nopar:cpn
4299       {
4300         @@ _ dotted _
4301         \int_use:N \l_@@_initial_i_int -
4302         \int_use:N \l_@@_initial_j_int
4303       }
4304       { }
4305     }
4306   }
4307 }
4308 }
```

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.

```

4309   \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4310   {
4311     { \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`.

```

4312     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4313     { \int_use:N \l_@@_final_i_int }
4314     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4315     { }
4316   }
4317 }

```

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

```

4318 \cs_new_protected:Npn \@@_open_shorten:
4319 {
4320   \bool_if:NT \l_@@_initial_open_bool
4321   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4322   \bool_if:NT \l_@@_final_open_bool
4323   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4324 }

```

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).

```

4325 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4326 {
4327   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4328   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4329   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4330   \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`.

```

4331 \seq_if_empty:NF \g_@@_submatrix_seq
4332 {
4333   \seq_map_inline:Nn \g_@@_submatrix_seq
4334   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4335 }
4336 }

```

`#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.

```

4337 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4338 {
4339   \if_int_compare:w
4340     #3 > #1
4341   \else:
4342     \if_int_compare:w #1 > #5
4343   \else:
4344     \if_int_compare:w #4 > #2
4345   \else:
4346     \if_int_compare:w #2 > #6
4347   \else:
4348     \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4349     \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4350     \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
4351     \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
4352   \fi:
4353   \fi:
4354   \fi:
4355 \fi:
4356 }

4357 \cs_new_protected:Npn \@@_set_initial_coords:
4358 {
4359   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4360   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4361 }
4362 \cs_new_protected:Npn \@@_set_final_coords:
4363 {
4364   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4365   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4366 }
4367 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4368 {
4369   \pgfpointanchor
4370   {
4371     \@@_env:
4372     - \int_use:N \l_@@_initial_i_int
4373     - \int_use:N \l_@@_initial_j_int
4374   }
4375   { #1 }
4376   \@@_set_initial_coords:
4377 }
4378 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4379 {
4380   \pgfpointanchor
4381   {
4382     \@@_env:
4383     - \int_use:N \l_@@_final_i_int
4384     - \int_use:N \l_@@_final_j_int
4385   }
4386   { #1 }
4387   \@@_set_final_coords:
4388 }

4389 \cs_new_protected:Npn \@@_open_x_initial_dim:
4390 {
4391   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4392   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4393   {

```

```

4394 \cs_if_exist:cT
4395 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4396 {
4397   \pgfpointanchor
4398   { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4399   { west }
4400   \dim_set:Nn \l_@@_x_initial_dim
4401   { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4402 }
4403 }

```

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

```

4404 \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4405 {
4406   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4407   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4408   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4409 }
4410 }
4411 \cs_new_protected:Npn \@@_open_x_final_dim:
4412 {
4413   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4414   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4415   {
4416     \cs_if_exist:cT
4417     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4418     {
4419       \pgfpointanchor
4420       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4421       { east }
4422       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4423       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4424     }
4425   }

```

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

```

4426 \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4427 {
4428   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4429   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4430   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4431 }
4432 }

```

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.

```

4433 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4434 {
4435   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4436   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4437   {
4438     \@@_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.

```

4439 \bool_if:NT \g_@@_aux_found_bool
4440 {
4441   \group_begin:
4442   \@@_open_shorten:
4443   \int_if_zero:nTF { #1 }
4444   { \color { nicematrix-first-row } }
4445   {

```

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.

```

4446         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4447         { \color { nicematrix-last-row } }
4448     }
4449     \keys_set:nn { nicematrix / xdots } { #3 }
4450     \@@_color:o \l_@@_xdots_color_tl
4451     \@@_actually_draw_Ldots:
4452 \group_end:
4453 }
4454 }
4455 }
```

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`.

```

4456 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4457 {
4458     \bool_if:NTF \l_@@_initial_open_bool
4459     {
4460         \@@_open_x_initial_dim:
4461         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4462         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4463     }
4464     { \@@_set_initial_coords_from_anchor:n { base~east } }
4465     \bool_if:NTF \l_@@_final_open_bool
4466     {
4467         \@@_open_x_final_dim:
4468         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4469         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4470     }
4471     { \@@_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.

```

4472     \bool_lazy_all:nTF
4473     {
4474         \l_@@_initial_open_bool
4475         \l_@@_final_open_bool
4476         { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4477     }
4478     {
4479         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4480         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4481     }
```

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

```

4482     {
4483         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4484         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4485     }
4486     \@@_draw_line:
4487 }
```

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.

```

4488 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4489 {
4490   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4491   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4492   {
4493     \@@_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.

```

4494     \bool_if:NT \g_@@_aux_found_bool
4495     {
4496       \group_begin:
4497       \@@_open_shorten:
4498       \int_if_zero:nTF { #1 }
4499       { \color { nicematrix-first-row } }
4500       {

```

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.

```

4501         \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4502         { \color { nicematrix-last-row } }
4503       }
4504       \keys_set:nn { nicematrix / xdots } { #3 }
4505       \@@_color:o \l_@@_xdots_color_tl
4506       \@@_actually_draw_Cdots:
4507     \group_end:
4508   }
4509 }
4510 }

```

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`.

```

4511 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4512 {
4513   \bool_if:NTF \l_@@_initial_open_bool
4514   { \@@_open_x_initial_dim: }
4515   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4516   \bool_if:NTF \l_@@_final_open_bool
4517   { \@@_open_x_final_dim: }
4518   { \@@_set_final_coords_from_anchor:n { mid-west } }
4519   \bool_lazy_and:nnTF
4520   { \l_@@_initial_open_bool }
4521   { \l_@@_final_open_bool }
4522   {
4523     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4524     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4525     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4526     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4527     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4528   }
4529   {
4530     \bool_if:NT \l_@@_initial_open_bool

```



```

4531         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4532         \bool_if:NT \l_@@_final_open_bool
4533         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4534     }
4535     \@@_draw_line:
4536 }
4537 \cs_new_protected:Npn \@@_open_y_initial_dim:
4538 {
4539     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4540     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4541     {
4542         \cs_if_exist:cT
4543         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4544         {
4545             \pgfpointanchor
4546             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4547             { north }
4548             \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4549             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4550         }
4551     }
4552     \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4553     {
4554         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4555         \dim_set:Nn \l_@@_y_initial_dim
4556         {
4557             \fp_to_dim:n
4558             {
4559                 \pgf@y
4560                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4561             }
4562         }
4563     }
4564 }
4565 \cs_new_protected:Npn \@@_open_y_final_dim:
4566 {
4567     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4568     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4569     {
4570         \cs_if_exist:cT
4571         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4572         {
4573             \pgfpointanchor
4574             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4575             { south }
4576             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4577             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4578         }
4579     }
4580     \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4581     {
4582         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4583         \dim_set:Nn \l_@@_y_final_dim
4584         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4585     }
4586 }

```

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.

```

4587 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4588 {
4589     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4590     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }

```

```

4591 {
4592   \@@_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.

```

4593   \bool_if:NT \g_@@_aux_found_bool
4594   {
4595     \group_begin:
4596     \@@_open_shorten:
4597     \int_if_zero:nTF { #2 }
4598       { \color { nicematrix-first-col } }
4599       {
4600         \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4601         { \color { nicematrix-last-col } }
4602       }
4603     \keys_set:nn { nicematrix / xdots } { #3 }
4604     \@@_color:o \l_@@_xdots_color_tl
4605     \bool_if:NTF \l_@@_Vbrace_bool
4606       { \@@_actually_draw_Vbrace: }
4607       { \@@_actually_draw_Vdots: }
4608     \group_end:
4609   }
4610 }
4611 }

```

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`.

```

4612 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4613 {
4614   \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4615   { \@@_actually_draw_Vdots_i: }
4616   { \@@_actually_draw_Vdots_ii: }
4617   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4618   \@@_draw_line:
4619 }

```

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

```

4620 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4621 {
4622   \@@_open_y_initial_dim:
4623   \@@_open_y_final_dim:
4624   \int_if_zero:nTF { \l_@@_initial_j_int }

```

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

```

4625 {
4626   \@@_qpoint:n { col - 1 }
4627   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4628   \dim_sub:Nn \l_@@_x_initial_dim
4629   { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4630 }
4631 {
4632   \bool_lazy_and:nnTF

```

```

4633     { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4634     {
4635         \int_compare_p:nNn
4636         { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4637     }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4638     {
4639         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4640         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4641         \dim_add:Nn \l_@@_x_initial_dim
4642         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4643     }

```

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

```

4644     {
4645         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4646         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4647         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4648         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4649     }
4650 }
4651 }

```

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.

```

4652 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4653 {
4654     \bool_set_false:N \l_tmpa_bool
4655     \bool_if:NF \l_@@_initial_open_bool
4656     {
4657         \bool_if:NF \l_@@_final_open_bool
4658         {
4659             \@@_set_initial_coords_from_anchor:n { south-west }
4660             \@@_set_final_coords_from_anchor:n { north-west }
4661             \bool_set:Nn \l_tmpa_bool
4662             {
4663                 \dim_compare_p:nNn
4664                 { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4665             }
4666         }
4667     }

```

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

```

4668     \bool_if:NTF \l_@@_initial_open_bool
4669     {
4670         \@@_open_y_initial_dim:
4671         \@@_set_final_coords_from_anchor:n { north }
4672         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4673     }
4674     {
4675         \@@_set_initial_coords_from_anchor:n { south }
4676         \bool_if:NTF \l_@@_final_open_bool
4677         { \@@_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.

```

4678     {
4679         \@@_set_final_coords_from_anchor:n { north }
4680         \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4681         {
4682             \dim_set:Nn \l_@@_x_initial_dim

```

```

4683         {
4684             \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4685             \l_@@_x_initial_dim \l_@@_x_final_dim
4686         }
4687     }
4688 }
4689 }
4690 }

```

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.`

```

4691 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4692 {
4693     \bool_if:NTF \l_@@_initial_open_bool
4694     { \@@_open_y_initial_dim: }
4695     { \@@_set_initial_coords_from_anchor:n { south } }
4696     \bool_if:NTF \l_@@_final_open_bool
4697     { \@@_open_y_final_dim: }
4698     { \@@_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.

```

4699     \int_if_zero:nTF { \l_@@_initial_j_int }
4700     {
4701         \@@_qpoint:n { col - 1 }
4702         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4703         \dim_sub:Nn \l_@@_x_initial_dim
4704         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4705     }

```

Elsewhere, the brace must be drawn left flush.

```

4706     {
4707         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4708         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4709         \dim_add:Nn \l_@@_x_initial_dim
4710         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4711     }

```

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

```

4712     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4713     \@@_draw_line:
4714 }

```

```

4715 \cs_new:Npn \@@_colsep:
4716 { \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.

```

4717 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4718 {
4719   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4720   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4721   {
4722     \@@_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.

```

4723     \bool_if:NT \g_@@_aux_found_bool
4724     {
4725       \group_begin:
4726       \@@_open_shorten:
4727       \keys_set:nn { nicematrix / xdots } { #3 }
4728       \@@_color:o \l_@@_xdots_color_tl
4729       \@@_actually_draw_Ddots:
4730       \group_end:
4731     }
4732   }
4733 }

```

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`.

```

4734 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4735 {
4736   \bool_if:NTF \l_@@_initial_open_bool
4737   {
4738     \@@_open_y_initial_dim:
4739     \@@_open_x_initial_dim:
4740   }
4741   { \@@_set_initial_coords_from_anchor:n { south~east } }
4742   \bool_if:NTF \l_@@_final_open_bool
4743   {
4744     \@@_open_x_final_dim:
4745     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4746   }
4747   { \@@_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.

```

4748   \bool_if:NT \l_@@_parallelize_diags_bool
4749   {
4750     \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).

```

4751     \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.

```

4752     {
4753         \dim_gset:Nn \g_@@_delta_x_one_dim
4754         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4755         \dim_gset:Nn \g_@@_delta_y_one_dim
4756         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4757     }

```

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`.

```

4758     {
4759         \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4760         {
4761             \dim_set:Nn \l_@@_y_final_dim
4762             {
4763                 \l_@@_y_initial_dim +
4764                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4765                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4766             }
4767         }
4768     }
4769 }
4770 \@@_draw_line:
4771 }

```

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.

```

4772 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4773 {
4774     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4775     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4776     {
4777         \@@_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.

```

4778         \bool_if:NT \g_@@_aux_found_bool
4779         {
4780             \group_begin:
4781             \@@_open_shorten:
4782             \keys_set:nn { nicematrix / xdots } { #3 }
4783             \@@_color:o \l_@@_xdots_color_tl
4784             \@@_actually_draw_Iddots:
4785             \group_end:
4786         }
4787     }
4788 }

```

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`.

```

4789 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4790 {
4791   \bool_if:NTF \l_@@_initial_open_bool
4792   {
4793     \@@_open_y_initial_dim:
4794     \@@_open_x_initial_dim:
4795   }
4796   { \@@_set_initial_coords_from_anchor:n { south-west } }
4797   \bool_if:NTF \l_@@_final_open_bool
4798   {
4799     \@@_open_y_final_dim:
4800     \@@_open_x_final_dim:
4801   }
4802   { \@@_set_final_coords_from_anchor:n { north-east } }
4803   \bool_if:NT \l_@@_parallelize_diags_bool
4804   {
4805     \int_gincr:N \g_@@_iddots_int
4806     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4807     {
4808       \dim_gset:Nn \g_@@_delta_x_two_dim
4809       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4810       \dim_gset:Nn \g_@@_delta_y_two_dim
4811       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4812     }
4813     {
4814       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4815       {
4816         \dim_set:Nn \l_@@_y_final_dim
4817         {
4818           \l_@@_y_initial_dim +
4819           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4820           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4821         }
4822       }
4823     }
4824   }
4825   \@@_draw_line:
4826 }

```

## 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`

```

4827 \cs_new_protected:Npn \@@_draw_line:
4828 {
4829   \pgfrememberpicturepositiononpagetrue
4830   \pgf@relevantforpicturesizefalse

```

```

4831 \bool_lazy_or:nnTF
4832 { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4833 { \l_@@_dotted_bool }
4834 { \@@_draw_standard_dotted_line: }
4835 { \@@_draw_unstandard_dotted_line: }
4836 }

```

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.

```

4837 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4838 {
4839   \begin { scope }
4840   \@@_draw_unstandard_dotted_line:o
4841   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4842 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4843 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4844 {
4845   \@@_draw_unstandard_dotted_line:nooo
4846   { #1 }
4847   \l_@@_xdots_up_tl
4848   \l_@@_xdots_down_tl
4849   \l_@@_xdots_middle_tl
4850 }
4851 \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.

```

4852 \hook_gput_code:nnn { begindocument } { . }
4853 {
4854   \IfPackageLoadedT { tikz }
4855   {
4856     \tikzset
4857     {
4858       @@_node_above / .style = { sloped , above } ,
4859       @@_node_below / .style = { sloped , below } ,
4860       @@_node_middle / .style =
4861       {
4862         sloped ,
4863         inner~sep = \c_@@_innersep_middle_dim
4864       }
4865     }
4866   }
4867 }

4868 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4869 {

```

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.

```

4870 \dim_zero_new:N \l_@@_l_dim
4871 \dim_set:Nn \l_@@_l_dim
4872 {
4873   \fp_to_dim:n
4874   {

```



```

4875         sqrt
4876         (
4877             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4878             +
4879             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4880         )
4881     }
4882 }

```

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.

```

4883     \dim_compare:nNt { \l_@@_l_dim } < { \c_@@_max_l_dim }
4884     {
4885         \dim_compare:nNt { \l_@@_l_dim } > { 1 pt }
4886         \@@_draw_unstandard_dotted_line_i:
4887     }

```

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

```

4888     \bool_if:NT \l_@@_xdots_h_labels_bool
4889     {
4890         \tikzset
4891         {
4892             @@_node_above / .style = { auto = left } ,
4893             @@_node_below / .style = { auto = right } ,
4894             @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4895         }
4896     }
4897     \tl_if_empty:nF { #4 }
4898     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4899     \dim_zero:N \l_tmpa_dim
4900     \dim_zero:N \l_tmpb_dim
4901     \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
4902     {

```

We test whether the brace is vertical or horizontal.

```

4903         \dim_compare:nNtF \l_@@_x_final_dim = \l_@@_x_initial_dim
4904         { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
4905         { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
4906     }
4907     {
4908         \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
4909         {
4910             \dim_compare:nNtF \l_@@_x_final_dim = \l_@@_x_initial_dim
4911             { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
4912             { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
4913         }
4914     }
4915     \use:e
4916     {
4917         \exp_not:N \begin { scope }
4918         [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
4919     }
4920     \draw
4921     [ #1 ]
4922     ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
4923     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4924     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4925     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4926     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4927     \end { scope }
4928     \end { scope }
4929 }

```

```

4930 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4931 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4932 {
4933   \dim_set:Nn \l_tmpa_dim
4934   {
4935     \l_@@_x_initial_dim
4936     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4937     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4938   }
4939   \dim_set:Nn \l_tmpb_dim
4940   {
4941     \l_@@_y_initial_dim
4942     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4943     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4944   }
4945   \dim_set:Nn \l_@@_tmpc_dim
4946   {
4947     \l_@@_x_final_dim
4948     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4949     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4950   }
4951   \dim_set:Nn \l_@@_tmpd_dim
4952   {
4953     \l_@@_y_final_dim
4954     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4955     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4956   }
4957   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4958   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4959   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4960   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4961 }

```

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

```

4962 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4963 {
4964   \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.

```

4965   \dim_zero_new:N \l_@@_l_dim
4966   \dim_set:Nn \l_@@_l_dim
4967   {
4968     \fp_to_dim:n
4969     {
4970       sqrt
4971       (
4972         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4973         +
4974         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4975       )
4976     }
4977   }

```

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.

```

4978   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4979   {
4980     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4981     { \@@_draw_standard_dotted_line_i: }

```

```

4982     }
4983   \group_end:
4984   \bool_lazy_all:nF
4985   {
4986     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4987     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4988     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4989   }
4990   { \@@_labels_standard_dotted_line: }
4991 }
4992 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4993 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4994 {

```

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

```

4995   \int_set:Nn \l_tmpa_int
4996   {
4997     \dim_ratio:nn
4998     {
4999       \l_@@_l_dim
5000       - \l_@@_xdots_shorten_start_dim
5001       - \l_@@_xdots_shorten_end_dim
5002     }
5003     { \l_@@_xdots_inter_dim }
5004   }

```

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

```

5005   \dim_set:Nn \l_tmpa_dim
5006   {
5007     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5008     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5009   }
5010   \dim_set:Nn \l_tmpb_dim
5011   {
5012     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5013     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
5014   }

```

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.

```

5015   \dim_gadd:Nn \l_@@_x_initial_dim
5016   {
5017     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
5018     \dim_ratio:nn
5019     {
5020       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5021       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5022     }
5023     { 2 \l_@@_l_dim }
5024   }
5025   \dim_gadd:Nn \l_@@_y_initial_dim
5026   {
5027     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
5028     \dim_ratio:nn
5029     {
5030       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
5031       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
5032     }
5033     { 2 \l_@@_l_dim }
5034   }
5035   \pgf@relevantforpicturesizefalse

```

```

5036 \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
5037 {
5038   \pgfpathcircle
5039   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5040   { \l_@@_xdots_radius_dim }
5041   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
5042   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
5043 }
5044 \pgfusepathqfill
5045 }

5046 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
5047 {
5048   \pgfscope
5049   \pgftransformshift
5050   {
5051     \pgfpointlineattime { 0.5 }
5052     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
5053     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
5054   }
5055   \fp_set:Nn \l_tmpa_fp
5056   {
5057     atand
5058     (
5059       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
5060       \l_@@_x_final_dim - \l_@@_x_initial_dim
5061     )
5062   }
5063   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
5064   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
5065   \tl_if_empty:NF \l_@@_xdots_middle_tl
5066   {
5067     \begin { pgfscope }
5068     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
5069     \pgfnode
5070     { rectangle }
5071     { center }
5072     {
5073       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5074       {
5075         $ % $
5076         \scriptstyle \l_@@_xdots_middle_tl
5077         $ % $
5078       }
5079     }
5080     { }
5081     {
5082       \pgfsetfillcolor { white }
5083       \pgfusepath { fill }
5084     }
5085     \end { pgfscope }
5086   }
5087   \tl_if_empty:NF \l_@@_xdots_up_tl
5088   {
5089     \pgfnode
5090     { rectangle }
5091     { south }
5092     {
5093       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5094       {
5095         $ % $
5096         \scriptstyle \l_@@_xdots_up_tl
5097         $ % $

```

```

5098         }
5099     }
5100     { }
5101     { \pgfusepath { } }
5102 }
5103 \tl_if_empty:NF \l_@@_xdots_down_tl
5104 {
5105     \pgfnode
5106     { rectangle }
5107     { north }
5108     {
5109         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
5110         {
5111             $ % $
5112             \scriptstyle \l_@@_xdots_down_tl
5113             $ % $
5114         }
5115     }
5116     { }
5117     { \pgfusepath { } }
5118 }
5119 \endpgfscope
5120 }

```

## 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.

```

5121 \hook_gput_code:nnn { begindocument } { . }
5122 {

```

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

```

5123     \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5124     \cs_new_protected:Npn \@@_Ldots:
5125     { \@@_collect_options:n { \@@_Ldots_i } }
5126     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5127     {
5128         \int_if_zero:nTF { \c@jCol }
5129         { \@@_error:nn { in~first~col } { \Ldots } }
5130         {
5131             \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5132             { \@@_error:nn { in~last~col } { \Ldots } }
5133             {
5134                 \@@_instruction_of_type:nnn { \c_false_bool } { \Ldots }
5135                 { #1 , down = #2 , up = #3 , middle = #4 }
5136             }
5137         }
5138         \bool_if:NF \l_@@_nullify_dots_bool
5139         { \phantom { \ensuremath { \@@_old_ldots: } } }
5140         \bool_gset_true:N \g_@@_empty_cell_bool
5141     }

```

```

5142 \cs_new_protected:Npn \@@_Cdots:
5143   { \@@_collect_options:n { \@@_Cdots_i } }
5144 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5145   {
5146     \int_if_zero:nTF { \c@jCol }
5147       { \@@_error:nn { in~first~col } { \Cdots } }
5148     {
5149       \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5150         { \@@_error:nn { in~last~col } { \Cdots } }
5151       {
5152         \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5153         { #1 , down = #2 , up = #3 , middle = #4 }
5154       }
5155     }
5156     \bool_if:NF \l_@@_nullify_dots_bool
5157       { \phantom { \ensuremath { \@@_old_cdots: } } }
5158     \bool_gset_true:N \g_@@_empty_cell_bool
5159   }

5160 \cs_new_protected:Npn \@@_Vdots:
5161   { \@@_collect_options:n { \@@_Vdots_i } }
5162 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5163   {
5164     \int_if_zero:nTF { \c@iRow }
5165       { \@@_error:nn { in~first~row } { \Vdots } }
5166     {
5167       \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5168         { \@@_error:nn { in~last~row } { \Vdots } }
5169       {
5170         \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5171         { #1 , down = #2 , up = #3 , middle = #4 }
5172       }
5173     }
5174     \bool_if:NF \l_@@_nullify_dots_bool
5175       { \phantom { \ensuremath { \@@_old_vdots: } } }
5176     \bool_gset_true:N \g_@@_empty_cell_bool
5177   }

5178 \cs_new_protected:Npn \@@_Ddots:
5179   { \@@_collect_options:n { \@@_Ddots_i } }
5180 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5181   {
5182     \int_case:nnF \c@iRow
5183     {
5184       0 { \@@_error:nn { in~first~row } { \Ddots } }
5185       \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5186     }
5187     {
5188       \int_case:nnF \c@jCol
5189       {
5190         0 { \@@_error:nn { in~first~col } { \Ddots } }
5191         \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5192       }
5193       {
5194         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5195         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5196         { #1 , down = #2 , up = #3 , middle = #4 }
5197       }
5198     }
5199   }
5200   \bool_if:NF \l_@@_nullify_dots_bool
5201     { \phantom { \ensuremath { \@@_old_ddots: } } }

```

```

5202     \bool_gset_true:N \g_@@_empty_cell_bool
5203 }

5204 \cs_new_protected:Npn \@@_Iddots:
5205 { \@@_collect_options:n { \@@_Iddots_i } }
5206 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5207 {
5208     \int_case:nnF \c@iRow
5209     {
5210         0 { \@@_error:nn { in~first~row } { \Iddots } }
5211         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5212     }
5213     {
5214         \int_case:nnF \c@jCol
5215         {
5216             0 { \@@_error:nn { in~first~col } { \Iddots } }
5217             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5218         }
5219         {
5220             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5221             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { \Iddots }
5222             { #1 , down = #2 , up = #3 , middle = #4 }
5223         }
5224     }
5225     \bool_if:NF \l_@@_nullify_dots_bool
5226     { \phantom { \ensuremath { \@@_old_iddots: } } }
5227     \bool_gset_true:N \g_@@_empty_cell_bool
5228 }
5229 }

```

End of the \AddToHook.

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

```

5230 \keys_define:nn { nicematrix / Ddots }
5231 {
5232     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5233     draw-first .default:n = true ,
5234     draw-first .value_forbidden:n = true
5235 }

```

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

```

5236 \cs_new_protected:Npn \@@_Hspace:
5237 {
5238     \bool_gset_true:N \g_@@_empty_cell_bool
5239     \hspace
5240 }

```

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`.

```

5241 \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`.

```

5242 \cs_new:Npn \@@_Hdotsfor:
5243 {
5244     \bool_lazy_and:nnTF
5245     { \int_if_zero_p:n { \c@jCol } }
5246     { \int_if_zero_p:n { \l_@@_first_col_int } }
5247     {

```

```

5248 \bool_if:NTF \g_@@_after_col_zero_bool
5249 {
5250   \multicolumn { 1 } { c } { }
5251   \@@_Hdotsfor_i:
5252 }
5253 { \@@_fatal:n { Hdotsfor~in~col~0 } }
5254 }
5255 {
5256   \multicolumn { 1 } { c } { }
5257   \@@_Hdotsfor_i:
5258 }
5259 }

```

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:`).

```

5260 \hook_gput_code:nnn { begindocument } { . }
5261 {

```

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

```

5262 \cs_new_protected:Npn \@@_Hdotsfor_i:
5263 { \@@_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`).

```

5264 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } { } } }
5265 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5266 {
5267   \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5268   {
5269     \@@_Hdotsfor:nnnn
5270     { \int_use:N \c@iRow }
5271     { \int_use:N \c@jCol }
5272     { #2 }
5273     {
5274       #1 , #3 ,
5275       down = \exp_not:n { #4 } ,
5276       up = \exp_not:n { #5 } ,
5277       middle = \exp_not:n { #6 }
5278     }
5279   }
5280   \prg_replicate:nn { #2 - 1 }
5281   {
5282     &
5283     \multicolumn { 1 } { c } { }
5284     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5285   }
5286 }
5287 }

```

```

5288 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5289 {
5290   \bool_set_false:N \l_@@_initial_open_bool
5291   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5292 \int_set:Nn \l_@@_initial_i_int { #1 }
5293 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5294 \int_compare:nNnTF { #2 } = { \c_one_int }
5295 {
5296   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5297   \bool_set_true:N \l_@@_initial_open_bool

```



```

5298 }
5299 {
5300   \cs_if_exist:cTF
5301   {
5302     pgf @ sh @ ns @ \@@_env:
5303     - \int_use:N \l_@@_initial_i_int
5304     - \int_eval:n { #2 - 1 }
5305   }
5306   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5307   {
5308     \int_set:Nn \l_@@_initial_j_int { #2 }
5309     \bool_set_true:N \l_@@_initial_open_bool
5310   }
5311 }
5312 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5313 {
5314   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5315   \bool_set_true:N \l_@@_final_open_bool
5316 }
5317 {
5318   \cs_if_exist:cTF
5319   {
5320     pgf @ sh @ ns @ \@@_env:
5321     - \int_use:N \l_@@_final_i_int
5322     - \int_eval:n { #2 + #3 }
5323   }
5324   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5325   {
5326     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5327     \bool_set_true:N \l_@@_final_open_bool
5328   }
5329 }
5330 \bool_if:NT \g_@@_aux_found_bool
5331 {
5332   \group_begin:
5333   \@@_open_shorten:
5334   \int_if_zero:nTF { #1 }
5335   { \color { nicematrix-first-row } }
5336   {
5337     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5338     { \color { nicematrix-last-row } }
5339   }
5340   \keys_set:nn { nicematrix / xdots } { #4 }
5341   \@@_color:o \l_@@_xdots_color_tl
5342   \@@_actually_draw_Ldots:
5343   \group_end:
5344 }

```

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).

```

5345   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5346   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5347 }

```

```

5348 \hook_gput_code:nnn { begindocument } { . }
5349 {
5350   \cs_new_protected:Npn \@@_Vdotsfor:
5351   { \@@_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`).

```

5352 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5353 \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5354 {
5355   \bool_gset_true:N \g_@@_empty_cell_bool
5356   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5357   {
5358     \@@_Vdotsfor:nnnn
5359     { \int_use:N \c@iRow }
5360     { \int_use:N \c@jCol }
5361     { #2 }
5362     {
5363       #1 , #3 ,
5364       down = \exp_not:n { #4 } ,
5365       up = \exp_not:n { #5 } ,
5366       middle = \exp_not:n { #6 }
5367     }
5368   }
5369 }
5370 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5371 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5372 {
5373   \bool_set_false:N \l_@@_initial_open_bool
5374   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5375   \int_set:Nn \l_@@_initial_j_int { #2 }
5376   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5377   \int_compare:nNnTF { #1 } = { \c_one_int }
5378   {
5379     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5380     \bool_set_true:N \l_@@_initial_open_bool
5381   }
5382   {
5383     \cs_if_exist:cTF
5384     {
5385       pgf @ sh @ ns @ \@@_env:
5386       - \int_eval:n { #1 - 1 }
5387       - \int_use:N \l_@@_initial_j_int
5388     }
5389     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5390     {
5391       \int_set:Nn \l_@@_initial_i_int { #1 }
5392       \bool_set_true:N \l_@@_initial_open_bool
5393     }
5394   }
5395   \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
5396   {
5397     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5398     \bool_set_true:N \l_@@_final_open_bool
5399   }
5400   {
5401     \cs_if_exist:cTF
5402     {
5403       pgf @ sh @ ns @ \@@_env:
5404       - \int_eval:n { #1 + #3 }
5405       - \int_use:N \l_@@_final_j_int
5406     }
5407     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }

```

```

5408     {
5409         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5410         \bool_set_true:N \l_@@_final_open_bool
5411     }
5412 }
5413 \bool_if:NT \g_@@_aux_found_bool
5414 {
5415     \group_begin:
5416     \@@_open_shorten:
5417     \int_if_zero:nTF { #2 }
5418     { \color { nicematrix-first-col } }
5419     {
5420         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5421         { \color { nicematrix-last-col } }
5422     }
5423     \keys_set:nn { nicematrix / xdots } { #4 }
5424     \@@_color:o \l_@@_xdots_color_tl
5425     \bool_if:NTF \l_@@_Vbrace_bool
5426     { \@@_actually_draw_Vbrace: }
5427     { \@@_actually_draw_Vdots: }
5428     \group_end:
5429 }

```

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).

```

5430     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5431     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5432 }

```

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

```

5433 \NewDocumentCommand \@@_rotate: { 0 { } }
5434 {
5435     \bool_gset_true:N \g_@@_rotate_bool
5436     \keys_set:nn { nicematrix / rotate } { #1 }
5437     \ignorespaces
5438 }
5439 \keys_define:nn { nicematrix / rotate }
5440 {
5441     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5442     c .value_forbidden:n = true ,
5443     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5444 }

```

## 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>

```

5445 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5446 {
5447   \tl_if_empty:nTF { #2 }
5448   { #1 }
5449   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5450 }
5451 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5452 { \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).

```

5453 \hook_gput_code:nnn { begindocument } { . }
5454 {
```

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

```

5455   \tl_set_rescan:Nnn \l_tmpa_tl { }
5456   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5457   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5458   {
5459     \group_begin:
5460     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5461     \@@_color:o \l_@@_xdots_color_tl
5462     \use:e
5463     {
5464       \@@_line_i:nn
5465       { \@@_double_int_eval:n #2 - \q_stop }
5466       { \@@_double_int_eval:n #3 - \q_stop }
5467     }
5468     \group_end:
5469   }
5470 }
5471 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5472 {
5473   \bool_set_false:N \l_@@_initial_open_bool
5474   \bool_set_false:N \l_@@_final_open_bool
5475   \bool_lazy_or:nnTF
5476   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5477   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5478   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```

5479   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5480 }
5481 \hook_gput_code:nnn { begindocument } { . }
5482 {
5483   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5484   {
```

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:.`

```

5485   \c_@@_pgfortikzpicture_tl
5486   \@@_draw_line_iii:nn { #1 } { #2 }
5487   \c_@@_endpgfortikzpicture_tl
```

---

<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`.

```

5488     }
5489 }

```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

5490 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5491 {
5492   \pgfrememberpicturepositiononpagetrue
5493   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5494   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5495   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5496   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5497   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5498   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5499   \@@_draw_line:
5500 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

## 20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```

5501 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNt { \c@iRow } < }
5502 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNf { \c@jCol } < }

```

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

```

5503 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5504 {
5505   \tl_gput_right:Ne \g_@@_row_style_tl
5506   {

```

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

```

5507   \exp_not:N
5508   \@@_if_row_less_than:nn
5509   { \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`).

```

5510     {
5511       \exp_not:N
5512       \@@_if_col_greater_than:nn
5513       { \int_eval:n { \c@jCol } }
5514       { \exp_not:n { #1 } \scan_stop: }
5515     }
5516   }
5517 }
5518 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5519 \keys_define:nn { nicematrix / RowStyle }
5520 {
5521   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5522   cell-space-top-limit .value_required:n = true ,
5523   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5524   cell-space-bottom-limit .value_required:n = true ,
5525   cell-space-limits .meta:n =
5526   {
5527     cell-space-top-limit = #1 ,
5528     cell-space-bottom-limit = #1 ,
5529   } ,
5530   color .tl_set:N = \l_@@_color_tl ,
5531   color .value_required:n = true ,
5532   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5533   bold .default:n = true ,
5534   nb-rows .code:n =
5535     \str_if_eq:eeTF { #1 } { * }
5536     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5537     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5538   nb-rows .value_required:n = true ,
5539   fill .tl_set:N = \l_@@_fill_tl ,
5540   fill .value_required:n = true ,

```

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

```

5541   opacity .tl_set:N = \l_@@_opacity_tl ,
5542   opacity .value_required:n = true ,
5543   rowcolor .tl_set:N = \l_@@_fill_tl ,
5544   rowcolor .value_required:n = true ,
5545   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5546   rounded-corners .default:n = 4 pt ,
5547   unknown .code:n =
5548     \@@_unknown_key:nn
5549     { nicematrix / RowStyle }
5550     { Unknown-key-for-RowStyle }
5551 }

```

```

5552 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5553 {
5554   \group_begin:
5555   \tl_clear:N \l_@@_fill_tl
5556   \tl_clear:N \l_@@_opacity_tl
5557   \tl_clear:N \l_@@_color_tl
5558   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5559   \dim_zero:N \l_@@_rounded_corners_dim
5560   \dim_zero:N \l_tmpa_dim
5561   \dim_zero:N \l_tmpb_dim
5562   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

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

```

5563   \tl_if_empty:NF \l_@@_fill_tl
5564   {
5565     \@@_add_opacity_to_fill:
5566     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5567     {

```

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

```

5568     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5569     { \int_use:N \c_iRow - \int_use:N \c_jCol }
5570     {
5571       \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5572       - *
5573     }
5574     { \dim_use:N \l_@@_rounded_corners_dim }
5575   }

```

```

5576     }
5577     \@@_put_in_row_style:n { \exp_not:n { #2 } }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5578     \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5579     {
5580         \@@_put_in_row_style:e
5581         {
5582             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5583             {

```

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

```

5584             \dim_set:Nn \l_@@_cell_space_top_limit_dim
5585             { \dim_use:N \l_tmpa_dim }
5586         }
5587     }
5588 }

```

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

```

5589     \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5590     {
5591         \@@_put_in_row_style:e
5592         {
5593             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5594             {
5595                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5596                 { \dim_use:N \l_tmpb_dim }
5597             }
5598         }
5599     }

```

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

```

5600     \tl_if_empty:NF \l_@@_color_tl
5601     {
5602         \@@_put_in_row_style:e
5603         {
5604             \mode_leave_vertical:
5605             \@@_color:n { \l_@@_color_tl }
5606         }
5607     }

```

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

```

5608     \bool_if:NT \l_@@_bold_row_style_bool
5609     {
5610         \@@_put_in_row_style:n
5611         {
5612             \exp_not:n
5613             {
5614                 \if_mode_math:
5615                 $ % $
5616                 \bfseries \boldmath
5617                 $ % $
5618             \else:
5619                 \bfseries \boldmath
5620             \fi:
5621         }
5622     }
5623 }
5624 \group_end:
5625 \g_@@_row_style_tl
5626 \ignorespaces
5627 }

```

The following commande must *not* be protected.

```

5628 \cs_new:Npn \@@_rounded_from_row:n #1

```

```

5629 {
5630   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

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

```

5631   { \int_eval:n { #1 } - 1 }
5632   {
5633     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5634     - \exp_not:n { \int_use:N \c@jCol }
5635   }
5636   { \dim_use:N \l_@@_rounded_corners_dim }
5637 }

```

## 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).

```

5638 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5639 {

```

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.

```

5640   \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`.

```

5641   \str_if_in:nnF { #1 } { !! }
5642   {
5643     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

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

```

5644     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5645   }
5646   \int_if_zero:nTF { \l_tmpa_int }

```

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

```

5647   {
5648     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5649     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5650   }

```



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

```

5651     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5652   }
5653   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5654   \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }

```

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

```

5655   \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5656   {
5657     \dim_compare:nNtT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5658     {

```

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

```

5659     \group_begin:
5660     \pgfsetcornersarced
5661     {
5662       \pgfpoint
5663       { \l_@@_tab_rounded_corners_dim }
5664       { \l_@@_tab_rounded_corners_dim }
5665     }

```

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.

```

5666     \bool_if:NTF \l_@@_hvlines_bool
5667     {
5668       \pgfpathrectanglecorners
5669       {
5670         \pgfpointadd
5671         { \@@_qpoint:n { row-1 } }
5672         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5673       }
5674       {
5675         \pgfpointadd
5676         {
5677           \@@_qpoint:n
5678           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5679         }
5680         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5681       }
5682     }
5683     {
5684       \pgfpathrectanglecorners
5685       { \@@_qpoint:n { row-1 } }
5686       {
5687         \pgfpointadd
5688         {
5689           \@@_qpoint:n
5690           { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5691         }
5692         { \pgfpoint \c_zero_dim \arrayrulewidth }
5693       }
5694     }
5695     \pgfusepath { clip }
5696     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5697   }
5698 }

```

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`).

```

5699 \cs_new_protected:Npn \@@_actually_color:
5700 {
5701   \pgfpicture
5702   \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.

```

5703   \@@_clip_with_rounded_corners:
5704   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5705   {
5706     \int_compare:nNnTF { ##1 } = { \c_one_int }
5707     {
5708       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5709       \use:c { g_@@_color _ 1 _tl }
5710       \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5711     }
5712     {
5713       \begin { pgfscope }
5714         \@@_color_opacity: ##2
5715         \use:c { g_@@_color _ ##1 _tl }
5716         \tl_gclear:c { g_@@_color _ ##1 _tl }
5717         \pgfusepath { fill }
5718       \end { pgfscope }
5719     }
5720   }
5721   \endpgfpicture
5722 }

```

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

```

5723 \cs_new_protected:Npn \@@_color_opacity:
5724 {
5725   \peek_meaning:NTF [
5726     { \@@_color_opacity:w }
5727     { \@@_color_opacity:w [ ] }
5728   }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5729 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5730 {
5731   \tl_clear:N \l_tmpa_tl
5732   \keys_set_known:nn { 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.

```

5733   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5734   \tl_if_empty:NTF \l_tmpb_tl
5735     { \@declaredcolor }
5736     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5737 }

```

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

```

5738 \keys_define:nn { nicematrix / color-opacity }
5739 {
5740   opacity .tl_set:N          = \l_tmpa_tl ,
5741   opacity .value_required:n = true
5742 }

```

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

```

5743 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5744 {
5745   \def \l_@@_rows_tl { #1 }
5746   \def \l_@@_cols_tl { #2 }
5747   \@@_cartesian_path:
5748 }

```

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

```

5749 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5750 {
5751   \tl_if_blank:nF { #2 }
5752   {
5753     \@@_add_to_colors_seq:en
5754     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5755     { \@@_cartesian_color:nn { #3 } { - } }
5756   }
5757 }

```

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

```

5758 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5759 {
5760   \tl_if_blank:nF { #2 }
5761   {
5762     \@@_add_to_colors_seq:en
5763     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5764     { \@@_cartesian_color:nn { - } { #3 } }
5765   }
5766 }

```

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

```

5767 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5768 {
5769   \tl_if_blank:nF { #2 }
5770   {
5771     \@@_add_to_colors_seq:en
5772     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5773     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5774   }
5775 }

```

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

```

5776 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5777 {
5778   \tl_if_blank:nF { #2 }
5779   {
5780     \@@_add_to_colors_seq:en
5781     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5782     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5783   }
5784 }

```

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

```

5785 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5786 {
5787   \@@_cut_on_hyphen:w #1 \q_stop
5788   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5789   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5790   \@@_cut_on_hyphen:w #2 \q_stop
5791   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5792   \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`.

```
5793 \@@_cartesian_path:n { #3 }
5794 }
```

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

```
5795 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5796 {
5797   \clist_map_inline:nn { #3 }
5798   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5799 }
```

```
5800 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5801 {
5802   \int_step_inline:nn { \c@iRow }
5803   {
5804     \int_step_inline:nn { \c@jCol }
5805     {
5806       \int_if_even:nTF { ####1 + ##1 }
5807       { \@@_cellcolor [ #1 ] { #2 } }
5808       { \@@_cellcolor [ #1 ] { #3 } }
5809       { ##1 - ####1 }
5810     }
5811   }
5812 }
```

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”.

```
5813 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5814 {
5815   \@@_rectanglecolor [ #1 ] { #2 }
5816   { 1 - 1 }
5817   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5818 }
```

```
5819 \keys_define:nn { nicematrix / rowcolors }
5820 {
5821   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5822   respect-blocks .default:n = true ,
5823   cols .tl_set:N = \l_@@_cols_tl ,
5824   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5825   restart .default:n = true ,
5826   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5827 }
```

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`.

```
5828 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5829 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5830 \group_begin:
5831 \seq_clear_new:N \l_@@_colors_seq
5832 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5833 \tl_clear_new:N \l_@@_cols_tl
5834 \tl_set:Nn \l_@@_cols_tl { - }
5835 \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).

```

5836 \int_zero_new:N \l_@@_color_int
5837 \int_set_eq:NN \l_@@_color_int \c_one_int
5838 \bool_if:NT \l_@@_respect_blocks_bool
5839 {

```

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`).

```

5840 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5841 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5842 { \@@_not_in_exterior_p:nnnnn ##1 }
5843 }
5844 \pgfpicture
5845 \pgf@relevantforpicturesizefalse

```

`#2` is the list of intervals of rows.

```

5846 \clist_map_inline:nn { #2 }
5847 {
5848 \tl_set:Nn \l_tmpa_tl { ##1 }
5849 \tl_if_in:NnTF \l_tmpa_tl { - }
5850 { \@@_cut_on_hyphen:w ##1 \q_stop }
5851 { \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.

```

5852 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5853 \int_set:Nn \l_@@_color_int
5854 { \bool_if:NNTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5855 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5856 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5857 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5858 \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).

```

5859 \bool_if:NT \l_@@_respect_blocks_bool
5860 {
5861 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5862 { \@@_intersect_our_row_p:nnnnn #####1 }
5863 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5864 }
5865 \tl_set:Nn \l_@@_rows_tl
5866 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5867 \tl_set:Nn \l_@@_color_tl
5868 {
5869 \@@_color_index:n
5870 {
5871 \int_mod:nn
5872 { \l_@@_color_int - 1 }
5873 { \seq_count:N \l_@@_colors_seq }
5874 + 1
5875 }

```

```

5876     }
5877     \tl_if_empty:NF \l_@@_color_tl
5878     {
5879         \@@_add_to_colors_seq:ee
5880         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5881         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5882     }
5883     \int_incr:N \l_@@_color_int
5884     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5885 }
5886 }
5887 \endpgfpicture
5888 \group_end:
5889 }

```

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.

```

5890 \cs_new:Npn \@@_color_index:n #1
5891 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5892     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5893     { \@@_color_index:n { #1 - 1 } }
5894     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5895 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5896 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5897 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5898 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5899 {
5900     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5901     { \int_set:Nn \l_tmpb_int { #3 } }
5902 }

```

```

5903 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5904 {
5905     \int_if_zero:nTF { #4 }
5906     { \prg_return_false: }
5907     {
5908         \int_compare:nNnTF { #2 } > { \c@jCol }
5909         { \prg_return_false: }
5910         { \prg_return_true: }
5911     }
5912 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5913 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5914 {
5915     \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5916     { \prg_return_false: }
5917     {
5918         \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5919         { \prg_return_false: }
5920         { \prg_return_true: }
5921     }
5922 }

```

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`).

```

5923 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5924 {
5925   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5926   {
5927     \bool_if:NTF \l_@@_nocolor_used_bool
5928     { \@@_cartesian_path_normal_ii: }
5929     {
5930       \clist_if_empty:NTF \l_@@_corners_cells_clist
5931       { \@@_cartesian_path_normal_i:n { #1 } }
5932       { \@@_cartesian_path_normal_ii: }
5933     }
5934   }
5935   { \@@_cartesian_path_normal_i:n { #1 } }
5936 }

```

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.

```

5937 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5938 {
5939   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5940   \clist_map_inline:Nn \l_@@_cols_tl
5941   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5942     \def \l_tmpa_tl { ##1 }
5943     \tl_if_in:NnTF \l_tmpa_tl { - }
5944     { \@@_cut_on_hyphen:w ##1 \q_stop }
5945     { \def \l_tmpb_tl { ##1 } }
5946     \tl_if_empty:NTF \l_tmpa_tl
5947     { \def \l_tmpa_tl { 1 } }
5948     {
5949       \str_if_eq:eeT { \l_tmpa_tl } { * }
5950       { \def \l_tmpa_tl { 1 } }
5951     }
5952     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5953     { \@@_error:n { Invalid~col~number } }
5954     \tl_if_empty:NTF \l_tmpb_tl
5955     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5956     {
5957       \str_if_eq:eeT { \l_tmpb_tl } { * }
5958       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5959     }
5960     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5961     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5962     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5963     \@@_qpoint:n { col - \l_tmpa_tl }
5964     \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5965     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5966     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5967     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5968     \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.

```

5969 \clist_map_inline:Nn \l_@@_rows_tl
5970 {
5971   \def \l_tmpa_tl { #####1 }
5972   \tl_if_in:NnTF \l_tmpa_tl { - }
5973     { \@@_cut_on_hyphen:w #####1 \q_stop }
5974     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5975   \tl_if_empty:NTF \l_tmpa_tl
5976     { \def \l_tmpa_tl { 1 } }
5977     {
5978       \str_if_eq:eeT { \l_tmpa_tl } { * }
5979       { \def \l_tmpa_tl { 1 } }
5980     }
5981   \tl_if_empty:NTF \l_tmpb_tl
5982     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5983     {
5984       \str_if_eq:eeT { \l_tmpb_tl } { * }
5985       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5986     }
5987   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5988     { \@@_error:n { Invalid-row-number } }
5989   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5990     { \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`.

```

5991 \cs_if_exist:cF
5992 { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5993 {
5994   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5995   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5996   \@@_qpoint:n { row - \l_tmpa_tl }
5997   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5998   \pgfpathrectanglecorners
5999     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6000     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6001 }
6002 }
6003 }
6004 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

6005 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
6006 {
6007   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6008   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6009 \clist_map_inline:Nn \l_@@_cols_tl
6010 {
6011   \@@_qpoint:n { col - ##1 }
6012   \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
6013     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
6014     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
6015   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
6016   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

6017 \clist_map_inline:Nn \l_@@_rows_tl
6018 {
6019   \@@_if_in_corner:nF { #####1 - ##1 }
6020   {
6021     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
6022     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
6023     \@@_qpoint:n { row - #####1 }

```



```

6024         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
6025         \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
6026         {
6027             \pgfpathrectanglecorners
6028             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6029             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6030         }
6031     }
6032 }
6033 }
6034 }

```

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`).

```

6035 \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

```

6036 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
6037 {
6038     \bool_set_true:N \l_@@_nocolor_used_bool
6039     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
6040     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

6041     \clist_map_inline:Nn \l_@@_rows_tl
6042     {
6043         \clist_map_inline:Nn \l_@@_cols_tl
6044         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
6045     }
6046 }

```

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.

```

6047 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
6048 {
6049     \clist_set_eq:NN \l_tmpa_clist #1
6050     \clist_clear:N #1
6051     \clist_map_inline:Nn \l_tmpa_clist
6052     {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

6053         \def \l_tmpa_tl { ##1 }
6054         \tl_if_in:NnTF \l_tmpa_tl { - }
6055         { \@@_cut_on_hyphen:w ##1 \q_stop }
6056         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
6057         \bool_lazy_or:nnT
6058         { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
6059         { \tl_if_blank_p:o \l_tmpa_tl }
6060         { \def \l_tmpa_tl { 1 } }
6061         \bool_lazy_or:nnT
6062         { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
6063         { \tl_if_blank_p:o \l_tmpb_tl }
6064         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6065         \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6066         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
6067         \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
6068         { \clist_put_right:Nn #1 { #####1 } }
6069     }
6070 }

```

The following command will be linked to `\cellcolor` in the tabular.

```
6071 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
6072 {
6073   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6074   {
```

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`).

```
6075     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
6076     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6077   }
6078   \ignorespaces
6079 }

6080 \NewDocumentCommand \@@_cellcolor_error { 0 { } m }
6081 { \@@_error:n { cellcolor~in~Block } }
6082 % \end{macrocode}
6083 %
6084 % \begin{macrocode}
6085 \NewDocumentCommand \@@_rowcolor_error { 0 { } m }
6086 { \@@_error:n { rowcolor~in~Block } }
6087 % \end{macrocode}
6088 %
6089 % \bigskip
6090 % The following command will be linked to |\rowcolor| in the tabular.
6091 % \begin{macrocode}
6092 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
6093 {
6094   \tl_gput_right:Ne \g_@@_pre_code_before_tl
6095   {
6096     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
6097     { \int_use:N \c@iRow - \int_use:N \c@jCol }
6098     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
6099   }
6100   \ignorespaces
6101 }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
6102 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
6103 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
6104 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
6105 {
```

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`.

```
6106   \seq_gclear:N \g_tmpa_seq
6107   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6108   { \@@_rowlistcolors_tabular:nnnn #1 }
6109   \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`.

```
6110   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
6111   {
```

```

6112     { \int_use:N \c@iRow }
6113     { \exp_not:n { #1 } }
6114     { \exp_not:n { #2 } }
6115     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
6116   }
6117   \ignorespaces
6118 }

```

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`).

```

6119 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
6120 {
6121   \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).

```

6122   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
6123   {
6124     \tl_gput_right:Ne \g_@@_pre_code_before_tl
6125     {
6126       \@@_rowlistcolors
6127       [ \exp_not:n { #2 } ]
6128       { #1 - \int_eval:n { \c@iRow - 1 } }
6129       { \exp_not:n { #3 } }
6130       [ \exp_not:n { #4 } ]
6131     }
6132   }
6133 }

```

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.

```

6134 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
6135 {
6136   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
6137   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
6138   \seq_gclear:N \g_@@_rowlistcolors_seq
6139 }

6140 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6141 {
6142   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6143   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6144 }

```

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.

```

6145 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
6146 {

```

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).

```

6147   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6148   {

```

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).

```

6149     \tl_gput_left:N\g_@@_pre_code_before_tl
6150     {
6151         \exp_not:N \columncolor [ #1 ]
6152         { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6153     }
6154 }
6155 }

6156 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6157 {
6158     \clist_map_inline:nn { #1 }
6159     {
6160         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6161         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6162         \columncolor { nocolor } { ##1 }
6163     }
6164 }

6165 \cs_new_protected:Npn \@@_EmptyRow:n #1
6166 {
6167     \clist_map_inline:nn { #1 }
6168     {
6169         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6170         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6171         \rowcolor { nocolor } { ##1 }
6172     }
6173 }

```

## 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`.

```

6174 \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`.

```

6175 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6176 {
6177     \int_if_zero:nTF { \l_@@_first_col_int }
6178     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6179     {
6180         \int_if_zero:nTF { \c@jCol }
6181         {
6182             \int_compare:nNnF { \c@iRow } = { -1 }
6183             {

```

```

6184         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6185         { #1 }
6186     }
6187 }
6188 { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6189 }
6190 }

```

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.

```

6191 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6192 {
6193     \int_if_zero:nF { \c@iRow }
6194     {
6195         \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6196         {
6197             \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6198             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6199         }
6200     }
6201 }

```

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`.

```

6202 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6203 {
6204     \IfPackageLoadedTF { tikz }
6205     {
6206         \IfPackageLoadedTF { booktabs }
6207         { #2 }
6208         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6209     }
6210     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6211 }
6212 \NewExpandableDocumentCommand { \@@_TopRule } { }
6213 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
6214 \cs_new:Npn \@@_TopRule_i:
6215 {
6216     \noalign \bgroup
6217     \peek_meaning:NTF [
6218     { \@@_TopRule_ii: }
6219     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6220 }
6221 \NewDocumentCommand \@@_TopRule_ii: { o }
6222 {
6223     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6224     {
6225         \@@_hline:n
6226         {
6227             position = \int_eval:n { \c@iRow + 1 } ,
6228             tikz =
6229             {
6230                 line-width = #1 ,
6231                 yshift = 0.25 \arrayrulewidth ,
6232                 shorten~< = - 0.5 \arrayrulewidth
6233             } ,
6234             total-width = #1
6235         }

```

```

6236     }
6237     \skip_vertical:n { \belowrulesep + #1 }
6238     \egroup
6239 }

6240 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6241 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }

6242 \cs_new:Npn \@@_BottomRule_i:
6243 {
6244     \noalign \bgroup
6245     \peek_meaning:NTF [
6246     { \@@_BottomRule_ii: }
6247     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6248 }

6249 \NewDocumentCommand \@@_BottomRule_ii: { o }
6250 {
6251     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6252     {
6253         \@@_hline:n
6254         {
6255             position = \int_eval:n { \c@iRow + 1 } ,
6256             tikz =
6257             {
6258                 line-width = #1 ,
6259                 yshift = 0.25 \arrayrulewidth ,
6260                 shorten~< = - 0.5 \arrayrulewidth
6261             } ,
6262             total-width = #1 ,
6263         }
6264     }
6265     \skip_vertical:N \aboverulesep
6266     \@@_create_row_node_i:
6267     \skip_vertical:n { #1 }
6268     \egroup
6269 }

6270 \NewExpandableDocumentCommand { \@@_MidRule } { }
6271 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }

6272 \cs_new:Npn \@@_MidRule_i:
6273 {
6274     \noalign \bgroup
6275     \peek_meaning:NTF [
6276     { \@@_MidRule_ii: }
6277     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6278 }

6279 \NewDocumentCommand \@@_MidRule_ii: { o }
6280 {
6281     \skip_vertical:N \aboverulesep
6282     \@@_create_row_node_i:
6283     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6284     {
6285         \@@_hline:n
6286         {
6287             position = \int_eval:n { \c@iRow + 1 } ,
6288             tikz =
6289             {
6290                 line-width = #1 ,
6291                 yshift = 0.25 \arrayrulewidth ,
6292                 shorten~< = - 0.5 \arrayrulewidth
6293             } ,
6294             total-width = #1 ,
6295         }
6296     }

```

```

6297     \skip_vertical:n { \belowrulesep + #1 }
6298     \egroup
6299 }

```

## 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.

```

6300 \keys_define:nn { nicematrix / Rules }
6301 {
6302     position .int_set:N = \l_@@_position_int ,
6303     position .value_required:n = true ,
6304     start .int_set:N = \l_@@_start_int ,
6305     end .code:n =
6306         \bool_lazy_or:nnTF
6307         { \tl_if_empty_p:n { #1 } }
6308         { \str_if_eq_p:ee { #1 } { last } }
6309         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6310         { \int_set:Nn \l_@@_end_int { #1 } }
6311 }

```

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.

```

6312 \keys_define:nn { nicematrix / RulesBis }
6313 {
6314     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6315     multiplicity .initial:n = 1 ,
6316     dotted .bool_set:N = \l_@@_dotted_bool ,
6317     dotted .initial:n = false ,
6318     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]`.

```

6319     color .code:n =
6320         \@@_set_CTarc:n { #1 }
6321         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6322     color .value_required:n = true ,
6323     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6324     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.

```

6325     tikz .code:n =
6326         \IfPackageLoadedTF { tikz }
6327         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6328         { \@@_error:n { tikz-without-tikz } } ,
6329     tikz .value_required:n = true ,
6330     total-width .dim_set:N = \l_@@_rule_width_dim ,
6331     total-width .value_required:n = true ,
6332     width .meta:n = { total-width = #1 } ,
6333     unknown .code:n =
6334         \@@_unknown_key:nn

```

```

6335         { nicematrix / RulesBis }
6336         { Unknown-key-for-RulesBis }
6337     }

```

## The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6338 \cs_new_protected:Npn \@@_vline:n #1
6339 {

```

The group is for the options.

```

6340     \group_begin:
6341     \int_set_eq:NN \l_@@_end_int \c@iRow
6342     \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).

```

6343     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6344     \@@_vline_i:
6345     \group_end:
6346 }

```

```

6347 \cs_new_protected:Npn \@@_vline_i:
6348 {

```

`\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`.

```

6349     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6350     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6351     \l_tmpa_tl
6352     {

```

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.

```

6353         \bool_gset_true:N \g_tmpa_bool
6354         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6355         { \@@_test_vline_in_block:nnnnn ##1 }
6356         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6357         { \@@_test_vline_in_block:nnnnn ##1 }
6358         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6359         { \@@_test_vline_in_stroken_block:nnnn ##1 }
6360         \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6361         \bool_if:NTF \g_tmpa_bool
6362         {
6363             \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.

```

6364         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6365     }
6366     {
6367         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6368         {
6369             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6370             \@@_vline_ii:
6371             \int_zero:N \l_@@_local_start_int
6372         }
6373     }
6374 }
6375 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6376 {

```



```

6377         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6378         \@@_vline_ii:
6379     }
6380 }

6381 \cs_new_protected:Npn \@@_test_in_corner_v:
6382 {
6383     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6384     {
6385         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6386         { \bool_set_false:N \g_tmpa_bool }
6387     }
6388     {
6389         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6390         {
6391             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6392             { \bool_set_false:N \g_tmpa_bool }
6393             {
6394                 \@@_if_in_corner:nT
6395                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6396                 { \bool_set_false:N \g_tmpa_bool }
6397             }
6398         }
6399     }
6400 }

6401 \cs_new_protected:Npn \@@_vline_ii:
6402 {
6403     \tl_clear:N \l_@@_tikz_rule_tl
6404     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6405     \bool_if:NTF \l_@@_dotted_bool
6406     { \@@_vline_iv: }
6407     {
6408         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6409         { \@@_vline_iii: }
6410         { \@@_vline_v: }
6411     }
6412 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6413 \cs_new_protected:Npn \@@_vline_iii:
6414 {
6415     \pgfpicture
6416     \pgfrememberpicturepositiononpagetrue
6417     \pgf@relevantforpicturesizefalse
6418     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6419     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6420     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6421     \dim_set:Nn \l_tmpb_dim
6422     {
6423         \pgf@x
6424         - 0.5 \l_@@_rule_width_dim
6425         +
6426         ( \arrayrulewidth * \l_@@_multiplicity_int
6427           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6428     }
6429     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6430     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6431     \bool_lazy_all:nT
6432     {
6433         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }

```

```

6434     { \cs_if_exist_p:N \CT@drsc@ }
6435     { ! \tl_if_blank_p:o \CT@drsc@ }
6436   }
6437   {
6438     \group_begin:
6439     \CT@drsc@
6440     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6441     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6442     \dim_set:Nn \l_@@_tmpd_dim
6443       {
6444         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6445         * ( \l_@@_multiplicity_int - 1 )
6446       }
6447     \pgfpathrectanglecorners
6448       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6449       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6450     \pgfusepath { fill }
6451     \group_end:
6452   }
6453   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6454   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6455   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6456   {
6457     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6458     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6459     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6460   }
6461   \CT@arc@
6462   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6463   \pgfsetrectcap
6464   \pgfusepathqstroke
6465   \endpgfpicture
6466 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6467 \cs_new_protected:Npn \@@_vline_iv:
6468 {
6469   \pgfpicture
6470   \pgfrememberpicturepositiononpagetrue
6471   \pgf@relevantforpicturesizefalse
6472   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6473   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6474   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6475   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6476   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6477   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6478   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6479   \CT@arc@
6480   \@@_draw_line:
6481   \endpgfpicture
6482 }

```

The following code is for the case when the user uses the key `tikz`.

```

6483 \cs_new_protected:Npn \@@_vline_v:
6484 {
6485   \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).

```

6486   \CT@arc@
6487   \tl_if_empty:NF \l_@@_rule_color_tl
6488   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }

```

```

6489 \pgfrememberpicturepositiononpagetrue
6490 \pgf@relevantforpicturesizefalse
6491 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6492 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6493 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6494 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6495 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6496 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6497 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6498 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6499 ( \l_tmpb_dim , \l_tmpa_dim ) --
6500 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6501 \end { tikzpicture }
6502 }

```

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).

```

6503 \cs_new_protected:Npn \@@_draw_vlines:
6504 {
6505   \int_step_inline:nnn
6506   {
6507     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6508     { 2 }
6509     { 1 }
6510   }
6511   {
6512     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6513     { \c@jCol }
6514     { \int_eval:n { \c@jCol + 1 } }
6515   }
6516   {
6517     \str_if_eq:eeF { \l_@@_vlines_clist } { all }
6518     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6519     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6520   }
6521 }

```

## 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}`.

```

6522 \cs_new_protected:Npn \@@_hline:n #1
6523 {

```

The group is for the options.

```

6524   \group_begin:
6525   \int_set_eq:NN \l_@@_end_int \c@jCol
6526   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6527   \@@_hline_i:
6528   \group_end:
6529 }
6530 \cs_new_protected:Npn \@@_hline_i:
6531 {

```

`\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`.

```

6532   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6533   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6534   \l_tmpb_tl
6535   {

```

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.

```
6536      \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6537      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6538      { \@@_test_hline_in_block:nnnnn ##1 }

6539      \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6540      { \@@_test_hline_in_block:nnnnn ##1 }

6541      \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6542      { \@@_test_hline_in_stroken_block:nnnn ##1 }

6543      \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6544      \bool_if:NTF \g_tmpa_bool
6545      {
6546          \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.

```
6547          { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6548      }
6549      {
6550          \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6551          {
6552              \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6553              \@@_hline_ii:
6554              \int_zero:N \l_@@_local_start_int
6555          }
6556      }
6557  }
6558  \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6559  {
6560      \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6561      \@@_hline_ii:
6562  }
6563 }
```

```
6564 \cs_new_protected:Npn \@@_test_in_corner_h:
6565 {
6566     \int_compare:nNnTF { \l_tmpa_tl } = { \c_iRow + 1 }
6567     {
6568         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6569         { \bool_set_false:N \g_tmpa_bool }
6570     }
6571     {
6572         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6573         {
6574             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6575             { \bool_set_false:N \g_tmpa_bool }
6576             {
6577                 \@@_if_in_corner:nT
6578                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6579                 { \bool_set_false:N \g_tmpa_bool }
6580             }
6581         }
6582     }
6583 }
```

```
6584 \cs_new_protected:Npn \@@_hline_ii:
6585 {
```

```

6586 \tl_clear:N \l_@@_tikz_rule_tl
6587 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6588 \bool_if:NTF \l_@@_dotted_bool
6589 { \@@_hline_iv: }
6590 {
6591   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6592   { \@@_hline_iii: }
6593   { \@@_hline_v: }
6594 }
6595 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6596 \cs_new_protected:Npn \@@_hline_iii:
6597 {
6598   \pgfpicture
6599   \pgfrememberpicturepositiononpagetrue
6600   \pgf@relevantforpicturesizefalse
6601   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6602   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6603   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6604   \dim_set:Nn \l_tmpb_dim
6605   {
6606     \pgf@y
6607     - 0.5 \l_@@_rule_width_dim
6608     +
6609     ( \arrayrulewidth * \l_@@_multiplicity_int
6610       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6611   }
6612   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6613   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6614   \bool_lazy_all:nT
6615   {
6616     { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6617     { \cs_if_exist_p:N \CT@drsc@ }
6618     { ! \tl_if_blank_p:o \CT@drsc@ }
6619   }
6620   {
6621     \group_begin:
6622     \CT@drsc@
6623     \dim_set:Nn \l_@@_tmpd_dim
6624     {
6625       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6626       * ( \l_@@_multiplicity_int - 1 )
6627     }
6628     \pgfpathrectanglecorners
6629     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6630     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6631     \pgfusepathqfill
6632     \group_end:
6633   }
6634   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6635   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6636   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6637   {
6638     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6639     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6640     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6641   }
6642   \CT@arc@
6643   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6644   \pgfsetrectcap
6645   \pgfusepathqstroke
6646   \endpgfpicture

```

6647 } }

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6648 \cs_new_protected:Npn \@@_hline_iv:
6649 {
6650   \pgfpicture
6651   \pgfrememberpicturepositiononpagetrue
6652   \pgf@relevantforpicturesizefalse
6653   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6654   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6655   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6656   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6657   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6658   \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6659   {
6660     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6661     \bool_if:NF \g_@@_delims_bool
6662     { \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.

```
6663   \tl_if_eq:NnF \g_@@_left_delim_tl (
6664     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6665   )
6666   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6667   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6668   \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }
6669   {
6670     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6671     \bool_if:NF \g_@@_delims_bool
6672     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6673     \tl_if_eq:NnF \g_@@_right_delim_tl )
6674     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6675   }
6676   \CT@arc@
6677   \@@_draw_line:
6678   \endpgfpicture
6679 }
```

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`).

```
6680 \cs_new_protected:Npn \@@_hline_v:
6681 {
6682   \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).

```

6683 \CT@arc@
6684 \tl_if_empty:NF \l_@@_rule_color_tl
6685 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6686 \pgfrememberpicturepositiononpagetrue
6687 \pgf@relevantforpicturesizefalse
6688 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6689 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6690 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6691 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6692 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6693 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6694 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6695 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6696 ( \l_tmpa_dim , \l_tmpb_dim ) --
6697 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6698 \end { tikzpicture }
6699 }

```

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).

```

6700 \cs_new_protected:Npn \@@_draw_hlines:
6701 {
6702   \int_step_inline:nnn
6703   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6704   {
6705     \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6706     { \c@iRow }
6707     { \int_eval:n { \c@iRow + 1 } }
6708   }
6709   {
6710     \str_if_eq:eeF { \l_@@_hlines_clist } { all }
6711     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6712     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6713   }
6714 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6715 \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.

```

6716 \cs_set:Npn \@@_Hline_i:n #1
6717 {
6718   \peek_remove_spaces:n
6719   {
6720     \peek_meaning:NTF \Hline
6721     { \@@_Hline_ii:nn { #1 + 1 } }
6722     { \@@_Hline_iii:n { #1 } }
6723   }
6724 }
6725 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6726 \cs_set:Npn \@@_Hline_iii:n #1
6727 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6728 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6729 {
6730   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6731   \skip_vertical:N \l_@@_rule_width_dim
6732   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

6733 {
6734   \@@_hline:n
6735   {
6736     multiplicity = #1 ,
6737     position = \int_eval:n { \c@iRow + 1 } ,
6738     total-width = \dim_use:N \l_@@_rule_width_dim ,
6739     #2
6740   }
6741 }
6742 \egroup
6743 }

```

### 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`).

```

6744 \cs_new_protected:Npn \@@_custom_line:n #1
6745 {
6746   \str_clear_new:N \l_@@_command_str
6747   \str_clear_new:N \l_@@_ccommand_str
6748   \str_clear_new:N \l_@@_letter_str
6749   \tl_clear_new:N \l_@@_other_keys_tl
6750   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
6751   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
6752   {
6753     \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }

```

We delete the last character which is a space.

```

6754     \str_set:Ne \l_@@_command_str
6755     { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
6756   }
6757   \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
6758   {
6759     \str_set:Ne \l_@@_ccommand_str
6760     { \str_tail:N \l_@@_ccommand_str }
6761     \str_set:Ne \l_@@_ccommand_str
6762     { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
6763   }

```

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.

```

6764   \bool_lazy_all:nTF
6765   {
6766     { \str_if_empty_p:N \l_@@_letter_str }
6767     { \str_if_empty_p:N \l_@@_command_str }
6768     { \str_if_empty_p:N \l_@@_ccommand_str }
6769   }
6770   { \@@_error:n { No~letter~and~no~command } }
6771   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6772 }
6773 \keys_define:nn { nicematrix / custom-line }
6774 {
6775   letter .str_set:N = \l_@@_letter_str ,
6776   letter .value_required:n = true ,
6777   command .str_set:N = \l_@@_command_str ,
6778   command .value_required:n = true ,
6779   ccommand .str_set:N = \l_@@_ccommand_str ,
6780   ccommand .value_required:n = true ,
6781 }

```



```

6782 \cs_new_protected:Npn \@@_custom_line_i:n #1
6783 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6784 \bool_set_false:N \l_@@_tikz_rule_bool
6785 \bool_set_false:N \l_@@_dotted_rule_bool
6786 \bool_set_false:N \l_@@_color_bool
6787 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6788 \bool_if:NT \l_@@_tikz_rule_bool
6789 {
6790   \IfPackageLoadedF { tikz }
6791   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6792   \bool_if:NT \l_@@_color_bool
6793   { \@@_error:n { color-in-custom-line-with-tikz } }
6794 }
6795 \bool_if:NT \l_@@_dotted_rule_bool
6796 {
6797   \int_compare:nNt { \l_@@_multiplicity_int } > { \c_one_int }
6798   { \@@_error:n { key-multiplicity-with-dotted } }
6799 }
6800 \str_if_empty:NF \l_@@_letter_str
6801 {
6802   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6803   { \@@_error:n { Several-letters } }
6804   {
6805     \tl_if_in:NoTF
6806     \c_@@_forbidden_letters_str
6807     \l_@@_letter_str
6808     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6809   }

```

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.

```

6810 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6811 { \@@_v_custom_line:nn { #1 } }
6812 }
6813 }
6814 }
6815 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6816 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6817 }
6818 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6819 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6820 \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.

```

6821 \keys_define:nn { nicematrix / custom-line-bis }
6822 {
6823   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6824   multiplicity .initial:n = 1 ,
6825   multiplicity .value_required:n = true ,
6826   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6827   color .value_required:n = true ,
6828   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6829   tikz .value_required:n = true ,
6830   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6831   dotted .value_forbidden:n = true ,

```

```

6832 total-width .code:n = { } ,
6833 total-width .value_required:n = true ,
6834 width .code:n = { } ,
6835 width .value_required:n = true ,
6836 sep-color .code:n = { } ,
6837 sep-color .value_required:n = true ,
6838 unknown .code:n =
6839   \@@_unknown_key:nn
6840   { nicematrix / custom-line-bis }
6841   { Unknown-key-for-custom-line }
6842 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6843 \bool_new:N \l_@@_dotted_rule_bool
6844 \bool_new:N \l_@@_tikz_rule_bool
6845 \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`.

```

6846 \keys_define:nn { nicematrix / custom-line-width }
6847 {
6848   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6849   multiplicity .initial:n = 1 ,
6850   multiplicity .value_required:n = true ,
6851   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6852   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6853   \bool_set_true:N \l_@@_total_width_bool ,
6854   total-width .value_required:n = true ,
6855   width .meta:n = { total-width = #1 } ,
6856   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6857 }

```

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`).

```

6858 \cs_new_protected:Npn \@@_h_custom_line:n #1
6859 {

```

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`).

```

6860   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6861   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6862 }

```

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`).

```

6863 \cs_new_protected:Npn \@@_c_custom_line:n #1
6864 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6865   \exp_args:Nc \NewExpandableDocumentCommand
6866   { nicematrix - \l_@@_ccommand_str }
6867   { 0 { } m }
6868   {
6869     \noalign
6870     {
6871       \@@_compute_rule_width:n { #1 , ##1 }
6872       \skip_vertical:n { \l_@@_rule_width_dim }

```

```

6873         \clist_map_inline:nn
6874         { ##2 }
6875         { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6876     }
6877 }
6878 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6879 }

```

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*.

```

6880 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6881 {
6882     \tl_if_in:nnTF { #2 } { - }
6883     { \@@_cut_on_hyphen:w #2 \q_stop }
6884     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6885     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6886     {
6887         \@@_hline:n
6888         {
6889             #1 ,
6890             start = \l_tmpa_tl ,
6891             end = \l_tmpb_tl ,
6892             position = \int_eval:n { \c@iRow + 1 } ,
6893             total-width = \dim_use:N \l_@@_rule_width_dim
6894         }
6895     }
6896 }
6897 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6898 {
6899     \bool_set_false:N \l_@@_tikz_rule_bool
6900     \bool_set_false:N \l_@@_total_width_bool
6901     \bool_set_false:N \l_@@_dotted_rule_bool
6902     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6903     \bool_if:NF \l_@@_total_width_bool
6904     {
6905         \bool_if:NTF \l_@@_dotted_rule_bool
6906         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6907         {
6908             \bool_if:NF \l_@@_tikz_rule_bool
6909             {
6910                 \dim_set:Nn \l_@@_rule_width_dim
6911                 {
6912                     \arrayrulewidth * \l_@@_multiplicity_int
6913                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6914                 }
6915             }
6916         }
6917     }
6918 }

```

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

```

6919 \cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
6920 {
6921     \str_if_eq:nnTF { #2 } { [ ]
6922     { \@@_v_custom_line_i:nw { #1 } [ ] }
6923     { \@@_v_custom_line_ii:nn { #2 } { #1 } }
6924 }
6925 \cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
6926 { \@@_v_custom_line:nn { #1 , #2 } }
6927 \cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
6928 {
6929     \@@_compute_rule_width:n { #2 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6930 \tl_gput_right:Ne \g_@@_array_preamble_tl
6931 { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6932 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6933 {
6934   \@@_vline:n
6935   {
6936     #2 ,
6937     position = \int_eval:n { \c@jCol + 1 } ,
6938     total-width = \dim_use:N \l_@@_rule_width_dim
6939   }
6940 }
6941 \@@_rec_preamble:n #1
6942 }
6943 \@@_custom_line:n
6944 { 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.

```

6945 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6946 {
6947   \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6948   {
6949     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6950     {
6951       \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6952       {
6953         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6954         { \bool_gset_false:N \g_tmpa_bool }
6955       }
6956     }
6957   }
6958 }

```

The same for vertical rules.

```

6959 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6960 {
6961   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6962   {
6963     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6964     {
6965       \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6966       {
6967         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6968         { \bool_gset_false:N \g_tmpa_bool }
6969       }
6970     }
6971   }
6972 }
6973 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6974 {
6975   \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6976   {
6977     \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6978     {
6979       \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6980       { \bool_gset_false:N \g_tmpa_bool }
6981     }

```

```

6982         \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6983         { \bool_gset_false:N \g_tmpa_bool }
6984     }
6985 }
6986 }
6987 }
6988 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6989 {
6990     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6991     {
6992         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6993         {
6994             \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6995             { \bool_gset_false:N \g_tmpa_bool }
6996             {
6997                 \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6998                 { \bool_gset_false:N \g_tmpa_bool }
6999             }
7000         }
7001     }
7002 }

```

## 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.

```

7003 \cs_new_protected:Npn \@@_compute_corners:
7004 {
7005     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
7006     { \@@_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`).

```

7007     \clist_clear:N \l_@@_corners_cells_clist
7008     \clist_map_inline:Nn \l_@@_corners_clist
7009     {
7010         \str_case:nnF { ##1 }
7011         {
7012             { NW }
7013             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
7014             { NE }
7015             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
7016             { SW }
7017             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
7018             { SE }
7019             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
7020         }
7021         { \@@_error:nn { bad~corner } { ##1 } }
7022     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

7023     \clist_if_empty:NF \l_@@_corners_cells_clist
7024     {

```

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.

```

7025     \tl_gput_right:Ne \g_@@_aux_tl
7026     {
7027         \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
7028         { \l_@@_corners_cells_clist }
7029     }
7030 }
7031 }

7032 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
7033 {
7034     \int_step_inline:nnn { #1 } { #3 }
7035     {
7036         \int_step_inline:nnn { #2 } { #4 }
7037         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
7038     }
7039 }

7040 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
7041 {
7042     \cs_if_exist:cTF
7043     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
7044     { \prg_return_true: }
7045     { \prg_return_false: }
7046 }

```

“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.

```

7047 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
7048 {

```

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.

```

7049     \bool_set_false:N \l_tmpa_bool
7050     \int_zero_new:N \l_@@_last_empty_row_int
7051     \int_set:Nn \l_@@_last_empty_row_int { #1 }
7052     \int_step_inline:nnnn { #1 } { #3 } { #5 }
7053     {
7054         \bool_lazy_or:nnTF
7055         {
7056             \cs_if_exist_p:c
7057             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
7058         }
7059         { \@@_if_in_block_p:nn { ##1 } { #2 } }
7060         { \bool_set_true:N \l_tmpa_bool }
7061     }
7062     \bool_if:NF \l_tmpa_bool
7063     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
7064 }
7065 }

```

Now, you determine the last empty cell in the row of number 1.

```

7066 \bool_set_false:N \l_tmpa_bool
7067 \int_zero_new:N \l_@@_last_empty_column_int
7068 \int_set:Nn \l_@@_last_empty_column_int { #2 }
7069 \int_step_inline:nnnn { #2 } { #4 } { #6 }
7070 {
7071   \bool_lazy_or:nnTF
7072   {
7073     \cs_if_exist_p:c
7074     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
7075   }
7076   { \@@_if_in_block_p:nn { #1 } { ##1 } }
7077   { \bool_set_true:N \l_tmpa_bool }
7078   {
7079     \bool_if:NF \l_tmpa_bool
7080     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
7081   }
7082 }

```

Now, we loop over the rows.

```

7083 \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
7084 {

```

We treat the row number ##1 with another loop.

```

7085 \bool_set_false:N \l_tmpa_bool
7086 \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
7087 {
7088   \bool_lazy_or:nnTF
7089   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
7090   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
7091   { \bool_set_true:N \l_tmpa_bool }
7092   {
7093     \bool_if:NF \l_tmpa_bool
7094     {
7095       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
7096       \clist_put_right:Nn
7097       \l_@@_corners_cells_clist
7098       { ##1 - #####1 }
7099       \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
7100     }
7101   }
7102 }
7103 }
7104 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

7105 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
7106 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NnT \l_@@_corners_cells_clist { #1 } ...`

## 24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

7107 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

7108 \keys_define:nn { nicematrix / NiceMatrixBlock }
7109 {
7110   auto-columns-width .code:n =
7111   {
7112     \bool_set_true:N \l_@@_block_auto_columns_width_bool
7113     \dim_gzero_new:N \g_@@_max_cell_width_dim
7114     \bool_set_true:N \l_@@_auto_columns_width_bool
7115   }
7116 }

7117 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
7118 {
7119   \int_gincr:N \g_@@_NiceMatrixBlock_int
7120   \dim_zero:N \l_@@_columns_width_dim
7121   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
7122   \bool_if:NT \l_@@_block_auto_columns_width_bool
7123   {
7124     \cs_if_exist:cT
7125     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
7126     {
7127       \dim_set:Nn \l_@@_columns_width_dim
7128       {
7129         \use:c
7130         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
7131       }
7132     }
7133   }
7134 }

```

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`).

```

7135 {
7136   \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.

```

7137   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
7138   {
7139     \bool_if:NT \l_@@_block_auto_columns_width_bool
7140     {
7141       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
7142       \iow_shipout:Ne \@mainaux
7143       {
7144         \cs_gset:cpn
7145         { @@ _ 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.

```

7146         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
7147       }
7148       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
7149     }
7150   }
7151   \ignorespacesafterend
7152 }

```



## 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).

```

7153 \cs_new_protected:Npn \@@_create_extra_nodes:
7154 {
7155   \bool_if:nTF \l_@@_medium_nodes_bool
7156   {
7157     \bool_if:NTF \l_@@_no_cell_nodes_bool
7158     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7159     {
7160       \bool_if:NTF \l_@@_large_nodes_bool
7161       \@@_create_medium_and_large_nodes:
7162       \@@_create_medium_nodes:
7163     }
7164   }
7165   {
7166     \bool_if:NT \l_@@_large_nodes_bool
7167     {
7168       \bool_if:NTF \l_@@_no_cell_nodes_bool
7169       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7170       \@@_create_large_nodes:
7171     }
7172   }
7173 }

```

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`.

```

7174 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7175 {
7176   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7177   {
7178     \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7179     \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7180     \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7181     \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7182   }
7183   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7184   {
7185     \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7186     \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7187     \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7188     \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7189   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7190 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7191 {
7192     \int_step_variable:nnNn
7193     \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.

```

7194 {
7195     \cs_if_exist:cT
7196     { 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`.

```

7197 {
7198     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7199     \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7200     { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7201     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7202     {
7203         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7204         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7205     }

```

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`.

```

7206     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7207     \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7208     { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7209     \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7210     {
7211         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7212         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7213     }
7214 }
7215 }
7216 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7217 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7218 {
7219     \dim_compare:nNnT
7220     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7221     {
7222         \@@_qpoint:n { row - \@@_i: - base }
7223         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7224         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7225     }
7226 }
7227 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7228 {
7229     \dim_compare:nNnT
7230     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7231     {
7232         \@@_qpoint:n { col - \@@_j: }
7233         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7234         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7235     }
7236 }
7237 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7238 \cs_new_protected:Npn \@@_create_medium_nodes:
7239 {
7240   \pgfpicture
7241   \pgfrememberpicturepositiononpagetrue
7242   \pgf@relevantforpicturesizefalse
7243   \@@_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”.

```

7244   \tl_set:Nn \l_@@_suffix_tl { -medium }
7245   \@@_create_nodes:
7246   \endpgfpicture
7247 }

```

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:`.

```

7248 \cs_new_protected:Npn \@@_create_large_nodes:
7249 {
7250   \pgfpicture
7251   \pgfrememberpicturepositiononpagetrue
7252   \pgf@relevantforpicturesizefalse
7253   \@@_computations_for_medium_nodes:
7254   \@@_computations_for_large_nodes:
7255   \tl_set:Nn \l_@@_suffix_tl { - large }
7256   \@@_create_nodes:
7257   \endpgfpicture
7258 }

7259 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7260 {
7261   \pgfpicture
7262   \pgfrememberpicturepositiononpagetrue
7263   \pgf@relevantforpicturesizefalse
7264   \@@_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”.

```

7265   \tl_set:Nn \l_@@_suffix_tl { - medium }
7266   \@@_create_nodes:
7267   \@@_computations_for_large_nodes:
7268   \tl_set:Nn \l_@@_suffix_tl { - large }
7269   \@@_create_nodes:
7270   \endpgfpicture
7271 }

```

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.

```

7272 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7273 {
7274   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7275   \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`.

```

7276   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7277   {
7278     \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:`

```

7279     {
7280     (
7281         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7282         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7283     )
7284     / 2
7285     }
7286     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7287     { l_@@_row _ \@@_i: _ min_dim }
7288 }
7289 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7290 {
7291     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7292     {
7293     (
7294         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7295         \dim_use:c
7296         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7297     )
7298     / 2
7299     }
7300     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7301     { l_@@_column _ \@@_j: _ max _ dim }
7302 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7303     \dim_sub:cn
7304     { l_@@_column _ 1 _ min _ dim }
7305     \l_@@_left_margin_dim
7306     \dim_add:cn
7307     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7308     \l_@@_right_margin_dim
7309 }

```

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).

```

7310 \cs_new_protected:Npn \@@_create_nodes:
7311 {
7312     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7313     {
7314         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7315         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7316         \@@_pgf_rect_node:nnnnn
7317         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7318         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7319         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7320         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7321         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7322         \str_if_empty:NF \l_@@_name_str
7323         {
7324             \pgfnodealias
7325             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7326             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7327         }
7328     }
7329 }
7330 \int_step_inline:nn { \c@iRow }

```

```

7331 {
7332   \pgfnodealias
7333   { \@@_env: - ##1 - last \l_@@_suffix_tl }
7334   { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7335 }
7336 \int_step_inline:nn { \c@jCol }
7337 {
7338   \pgfnodealias
7339   { \@@_env: - last - ##1 \l_@@_suffix_tl }
7340   { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7341 }
7342 \pgfnodealias % added 2025-04-05
7343 { \@@_env: - last - last \l_@@_suffix_tl }
7344 { \@@_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  $n$ .

```

7345   \seq_map_pairwise_function:NNN
7346   \g_@@_multicolumn_cells_seq
7347   \g_@@_multicolumn_sizes_seq
7348   \@@_node_for_multicolumn:nn
7349 }

7350 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7351 {
7352   \cs_set_nopar:Npn \@@_i: { #1 }
7353   \cs_set_nopar:Npn \@@_j: { #2 }
7354 }

```

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”).

```

7355 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7356 {
7357   \@@_extract_coords_values: #1 \q_stop
7358   \@@_pgf_rect_node:nnnnn
7359   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7360   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7361   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7362   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7363   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7364   \str_if_empty:NF \l_@@_name_str
7365   {
7366     \pgfnodealias
7367     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7368     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7369   }
7370 }

```

## 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).

```

7371 \keys_define:nn { nicematrix / Block / FirstPass }
7372 {
7373   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7374     \bool_set_true:N \l_@@_p_block_bool ,
7375   j .value_forbidden:n = true ,
7376   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7377   l .value_forbidden:n = true ,
7378   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7379   r .value_forbidden:n = true ,
7380   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7381   c .value_forbidden:n = true ,
7382   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7383   L .value_forbidden:n = true ,
7384   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7385   R .value_forbidden:n = true ,
7386   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7387   C .value_forbidden:n = true ,
7388   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7389   t .value_forbidden:n = true ,
7390   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7391   T .value_forbidden:n = true ,
7392   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7393   b .value_forbidden:n = true ,
7394   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7395   B .value_forbidden:n = true ,
7396   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7397   m .value_forbidden:n = true ,
7398   v-center .meta:n = m ,
7399   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7400   p .value_forbidden:n = true ,
7401   color .code:n =
7402     \@@_color:n { #1 }
7403     \tl_set_rescan:Nnn
7404       \l_@@_draw_tl
7405       { \char_set_catcode_other:N ! }
7406       { #1 } ,
7407   color .value_required:n = true ,
7408   respect-arraystretch .code:n =
7409     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7410   respect-arraystretch .value_forbidden:n = true ,
7411 }

```

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.

```

7412 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7413 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7414 {

```

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).

```

7415   \tl_if_blank:nTF { #2 }
7416   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7417   {
7418     \tl_if_in:nnTF { #2 } { - }
7419     {
7420       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7421       \@@_Block_i_czech:w \@@_Block_i:w
7422       #2 \q_stop
7423     }
7424     {
7425       \@@_error:nn { Bad~argument~for~Block } { #2 }

```

```

7426         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7427     }
7428 }
7429 { #1 } { #3 } { #4 }
7430 \ignorespaces
7431 }

```

With the following construction, we extract the values of  $i$  and  $j$  in the first mandatory argument of the command.

```

7432 \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`.

```

7433 {
7434   \char_set_catcode_active:N -
7435   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7436 }

```

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.

```

7437 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7438 {

```

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).

```

7439   \bool_lazy_or:nnTF
7440     { \tl_if_blank_p:n { #1 } }
7441     { \str_if_eq_p:ee { * } { #1 } }
7442     { \int_set:Nn \l_tmpa_int { 100 } }
7443     { \int_set:Nn \l_tmpa_int { #1 } }
7444   \bool_lazy_or:nnTF
7445     { \tl_if_blank_p:n { #2 } }
7446     { \str_if_eq_p:ee { * } { #2 } }
7447     { \int_set:Nn \l_tmpb_int { 100 } }
7448     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7449   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7450   {
7451     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7452       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7453       { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7454   }
7455   { \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.

```

7456   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7457   \tl_set:Ne \l_tmpa_tl
7458   {
7459     { \int_use:N \c@iRow }
7460     { \int_use:N \c@jCol }
7461     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7462     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7463   }

```

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).

```

7464     \bool_set_false:N \l_tmpa_bool
7465     \bool_if:NT \l_@@_amp_in_blocks_bool
\tl_if_in:nnT is slightly faster than \str_if_in:nnT.
7466     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7467     \bool_case:nF
7468     {
7469         \l_tmpa_bool                                { \@@_Block_vii:eennn }
7470         \l_@@_p_block_bool                            { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right away in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7471         \l_@@_X_bool                                { \@@_Block_v:eennn }
7472         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7473         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7474         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7475     }
7476     { \@@_Block_v:eennn }
7477     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7478 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

**#1** is *i* (the number of rows of the block), **#2** is *j* (the number of columns of the block), **#3** is the list of `key=values` pairs, **#4** are the tokens to put before the potential math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

7479 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7480 {
7481     \int_gincr:N \g_@@_block_box_int
7482     \cs_set_eq:NN \cellcolor \@@_cellcolor_error
7483     \cs_set_eq:NN \rowcolor \@@_rowcolor_error
7484     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7485     {
7486         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7487         {
7488             \@@_actually_diagbox:nnnnnn
7489             { \int_use:N \c@iRow }
7490             { \int_use:N \c@jCol }
7491             { \int_eval:n { \c@iRow + #1 - 1 } }
7492             { \int_eval:n { \c@jCol + #2 - 1 } }
7493             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7494             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7495         }
7496     }
7497     \box_gclear_new:c
7498     { g_@@_block_box_int _int_use:N \g_@@_block_box_int _box }

```



Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7499   \hbox_gset:cn
7500   { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7501   {

```

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`).

```

7502   \tl_if_empty:NTF \l_@@_color_tl
7503   { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7504   { \@@_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`.

```

7505   \int_compare:nNnT { #1 } = { \c_one_int }
7506   {
7507       \int_if_zero:nTF { \c@iRow }
7508       {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & & -4 & 5 & \\
3 & & -4 & 5 & & -6 & \\
-4 & 5 & & -6 & 7 & & \\
5 & & -6 & 7 & & -8 & \\
\end{bNiceMatrix}$

```

```

7509   \cs_set_eq:NN \Block \@@_NullBlock:
7510   \l_@@_code_for_first_row_tl
7511   }
7512   {
7513       \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7514       {
7515           \cs_set_eq:NN \Block \@@_NullBlock:
7516           \l_@@_code_for_last_row_tl
7517       }
7518   }
7519   \g_@@_row_style_tl
7520   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7521   \@@_reset_arraystretch:
7522   \dim_zero:N \extrarowheight

```

**#4** is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7523   #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.).

```
7524 \@@_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}`.

```
7525 \bool_if:NTF \l_@@_tabular_bool
7526 {
7527   \bool_lazy_all:nTF
7528   {
7529     { \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.

```
7530 {
7531   ! \dim_compare_p:nNn
7532   { \l_@@_col_width_dim } < { \c_zero_dim }
7533 }
7534 { ! \g_@@_rotate_bool }
7535 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7536 {
7537   \use:e
7538   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7539 \exp_not:N \begin { minipage }
7540 [ \str_lowercase:f \l_@@_vpos_block_str ]
7541 { \l_@@_col_width_dim }
7542 \str_case:on \l_@@_hpos_block_str
7543 { c \centering r \raggedleft l \raggedright }
7544 }
7545 #5
7546 \end { minipage }
7547 }
```

In the other cases, we use a `{tabular}`.

```
7548 {
7549   \use:e
7550   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
7551 \exp_not:N \begin { tabular }
7552 [ \str_lowercase:f \l_@@_vpos_block_str ]
7553 { @ { } \l_@@_hpos_block_str @ { } }
7554 }
7555 #5
7556 \end { tabular }
7557 }
7558 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7559 {
7560   $ % $
7561   \use:e
7562   {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```

7563         \exp_not:N \begin { array }
7564         [ \str_lowercase:f \l_@@_vpos_block_str ]
7565         { @ { } \l_@@_hpos_block_str @ { } }
7566     }
7567     #5
7568     \end { array }
7569     $ % $
7570 }
7571 }
```

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).

```

7572     \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.

```

7573     \int_compare:nNnT { #2 } = { \c_one_int }
7574     {
7575         \dim_gset:Nn \g_@@_blocks_wd_dim
7576         {
7577             \dim_max:nn
7578             { \g_@@_blocks_wd_dim }
7579             {
7580                 \box_wd:c
7581                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7582             }
7583         }
7584     }
```

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.

```

7585     \int_compare:nNnT { #1 } = { \c_one_int }
7586     {
7587         \bool_lazy_any:nT
7588         {
7589             { \str_if_empty_p:N \l_@@_vpos_block_str }
7590             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
7591             { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
7592         }
7593         { \@@_adjust_blocks_ht_dp: }
7594     }
7595     \seq_gput_right:Ne \g_@@_blocks_seq
7596     {
7597         \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.

```

7598     {
7599         \exp_not:n { #3 } ,
7600         \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```

7601     \bool_if:NT \g_@@_rotate_bool
7602     {
7603         \bool_if:NTF \g_@@_rotate_c_bool
7604         { m }
7605         {
```

```

7606         \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7607         { T }
7608     }
7609 }
7610 }
7611 {
7612     \box_use_drop:c
7613     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7614 }
7615 }
7616 \bool_set_false:N \g_@@_rotate_c_bool
7617 }

7618 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7619 {
7620     \dim_gset:Nn \g_@@_blocks_ht_dim
7621     {
7622         \dim_max:nn
7623         { \g_@@_blocks_ht_dim }
7624         {
7625             \box_ht:c
7626             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7627         }
7628     }
7629     \dim_gset:Nn \g_@@_blocks_dp_dim
7630     {
7631         \dim_max:nn
7632         { \g_@@_blocks_dp_dim }
7633         {
7634             \box_dp:c
7635             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7636         }
7637     }
7638 }

7639 \cs_new:Npn \@@_adjust_hpos_rotate:
7640 {
7641     \bool_if:NT \g_@@_rotate_bool
7642     {
7643         \str_set:Ne \l_@@_hpos_block_str
7644         {
7645             \bool_if:NTF \g_@@_rotate_c_bool
7646             { c }
7647             {
7648                 \str_case:onF \l_@@_vpos_block_str
7649                 { b l B l t r T r }
7650                 {
7651                     \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7652                     { r }
7653                     { l }
7654                 }
7655             }
7656         }
7657     }
7658 }

7659 \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*.

```

7660 \cs_new_protected:Npn \@@_rotate_box_of_block:
7661 {
7662     \box_grotate:cn

```

```

7663     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7664     { 90 }
7665 \int_compare:nNt { \c@iRow } = { \l_@@_last_row_int }
7666 {
7667     \vbox_gset_top:cn
7668     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7669     {
7670         \skip_vertical:n { 0.8 ex }
7671         \box_use:c
7672         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7673     }
7674 }
7675 \bool_if:NT \g_@@_rotate_c_bool
7676 {
7677     \hbox_gset:cn
7678     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7679     {
7680         $ % $
7681         \vcenter
7682         {
7683             \box_use:c
7684             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7685         }
7686         $ % $
7687     }
7688 }
7689 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right away in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#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.

```

7690 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7691 {
7692     \seq_gput_right:Ne \g_@@_blocks_seq
7693     {
7694         \l_tmpa_tl
7695         { \exp_not:n { #3 } }
7696         {
7697             \bool_if:NTF \l_@@_tabular_bool
7698             {
7699                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7700     \@@_reset_arraystretch:
7701     \exp_not:n
7702     {
7703         \dim_zero:N \extrarowheight
7704         #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).

```

7705     \tag_if_active:T { \tag_stop:n { table } }
7706     \use:e
7707     {
7708         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7709         { @ { } \l_@@_hpos_block_str @ { } }

```

```

7710         }
7711         #5
7712     \end { tabular }
7713 }
7714 \group_end:
7715 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7716     {
7717     \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7718         \@@_reset_arraystretch:
7719         \exp_not:n
7720         {
7721             \dim_zero:N \extrarowheight
7722             #4
7723             $ % $
7724             \use:e
7725             {
7726                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7727                 { @ { } \l_@@_hpos_block_str @ { } }
7728             }
7729             #5
7730             \end { array }
7731             $ % $
7732         }
7733     \group_end:
7734 }
7735 }
7736 }
7737 }
7738 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a \Block which uses the key p.

```

7739 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7740 {
7741     \seq_gput_right:Ne \g_@@_blocks_seq
7742     {
7743         \l_tmpa_tl
7744         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7745         { { \exp_not:n { #4 #5 } } }
7746     }
7747 }
7748 \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.

```

7749 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7750 {
7751     \seq_gput_right:Ne \g_@@_blocks_seq
7752     {
7753         \l_tmpa_tl
7754         { \exp_not:n { #3 } }
7755         { \exp_not:n { #4 #5 } }
7756     }
7757 }
7758 \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).

```

7759 \keys_define:nn { nicematrix / Block / SecondPass }
7760 {
7761   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7762   ampersand-in-blocks .default:n = true ,
7763   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7764   tikz .code:n =
7765     \IfPackageLoadedTF { tikz }
7766     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7767     { \@@_error:n { tikz~key~without~tikz } } ,
7768   tikz .value_required:n = true ,
7769   fill .code:n =
7770     \tl_set_rescan:Nnn
7771     \l_@@_fill_tl
7772     { \char_set_catcode_other:N ! }
7773     { #1 } ,
7774   fill .value_required:n = true ,

```

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

```

7775   opacity .tl_set:N = \l_@@_opacity_tl ,
7776   opacity .value_required:n = true ,
7777   draw .code:n =
7778     \tl_set_rescan:Nnn
7779     \l_@@_draw_tl
7780     { \char_set_catcode_other:N ! }
7781     { #1 } ,
7782   draw .default:n = default ,
7783   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7784   rounded-corners .default:n = 4 pt ,
7785   color .code:n =
7786     \@@_color:n { #1 }
7787     \tl_set_rescan:Nnn
7788     \l_@@_draw_tl
7789     { \char_set_catcode_other:N ! }
7790     { #1 } ,
7791   borders .clist_set:N = \l_@@_borders_clist ,
7792   borders .value_required:n = true ,
7793   hvlines .meta:n = { vlines , hlines } ,
7794   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7795   vlines .default:n = true ,
7796   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7797   hlines .default:n = true ,
7798   line-width .dim_set:N = \l_@@_line_width_dim ,
7799   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7800   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7801             \bool_set_true:N \l_@@_p_block_bool ,
7802   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7803   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7804   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7805   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7806             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7807   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7808             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7809   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7810             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7811   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7812   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7813   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7814   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7815   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7816   m .value_forbidden:n = true ,
7817   v-center .meta:n = m ,

```

```

7818 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7819 p .value_forbidden:n = true ,
7820 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7821 name .value_required:n = true ,
7822 name .initial:n = ,
7823 respect-arraystretch .code:n =
7824   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7825 respect-arraystretch .value_forbidden:n = true ,
7826 transparent .bool_set:N = \l_@@_transparent_bool ,
7827 transparent .default:n = true ,
7828 transparent .initial:n = false ,
7829 unknown .code:n =
7830   \@@_unknown_key:nn
7831     { nicematrix / Block / SecondPass }
7832     { Unknown-key-for-Block }
7833 }

```

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.

```

7834 \cs_new_protected:Npn \@@_draw_blocks:
7835 {
7836   \bool_if:NTF \c_@@_revtex_bool
7837     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7838     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7839   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7840 }
7841 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7842 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7843   \int_zero:N \l_@@_last_row_int
7844   \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”).

```

7845   \int_compare:nNnTF { #3 } > { 98 }
7846     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7847     { \int_set:Nn \l_@@_last_row_int { #3 } }
7848   \int_compare:nNnTF { #4 } > { 98 }
7849     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7850     { \int_set:Nn \l_@@_last_col_int { #4 } }
7851   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7852     {
7853       \bool_lazy_and:nnTF
7854         { \l_@@_preamble_bool }
7855         {
7856           \int_compare_p:n
7857             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7858         }
7859       {
7860         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7861         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7862         \@@_msg_redirect_name:nn { columns-not-used } { none }
7863       }
7864     } { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7865 }

```



```

7866 {
7867   \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7868   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7869   {
7870     \@@_Block_v:nneenn
7871     { #1 }
7872     { #2 }
7873     { \int_use:N \l_@@_last_row_int }
7874     { \int_use:N \l_@@_last_col_int }
7875     { #5 }
7876     { #6 }
7877   }
7878 }
7879 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of `key=value` options; #6 is the label (content) of the block.

```

7880 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7881 {

```

The group is for the keys.

```

7882   \group_begin:
7883   \int_compare:nNnT { #1 } = { #3 }
7884   { \str_set:Nn \l_@@_vpos_block_str { t } }
7885   \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`.

```

7886   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7887   \bool_lazy_and:nnT
7888   { \l_@@_vlines_block_bool }
7889   { ! \l_@@_ampersand_bool }
7890   {
7891     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7892     {
7893       \@@_vlines_block:nnn
7894       { \exp_not:n { #5 } }
7895       { #1 - #2 }
7896       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7897     }
7898   }
7899   \bool_if:NT \l_@@_hlines_block_bool
7900   {
7901     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7902     {
7903       \@@_hlines_block:nnn
7904       { \exp_not:n { #5 } }
7905       { #1 - #2 }
7906       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7907     }
7908   }
7909   \bool_if:NF \l_@@_transparent_bool
7910   {
7911     \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7912     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7913     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7914     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7915   }
7916 }

```

```

7917 \tl_if_empty:NF \l_@@_draw_tl
7918 {
7919   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7920   { \@@_error:n { hlines~with~color } }
7921   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7922   {
7923     \@@_stroke_block:nnn
7924
7925     { \exp_not:n { #5 } }
7926     { #1 - #2 }
7927     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7928   }
7929   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7930   { { #1 } { #2 } { #3 } { #4 } }
7931 }
7932 \clist_if_empty:NF \l_@@_borders_clist
7933 {
7934   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7935   {
7936     \@@_stroke_borders_block:nnn
7937     { \exp_not:n { #5 } }
7938     { #1 - #2 }
7939     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7940   }
7941 }
7942 \tl_if_empty:NF \l_@@_fill_tl
7943 {
7944   \@@_add_opacity_to_fill:
7945   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7946   {
7947     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7948     { #1 - #2 }
7949     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7950     { \dim_use:N \l_@@_rounded_corners_dim }
7951   }
7952 }
7953 \seq_if_empty:NF \l_@@_tikz_seq
7954 {
7955   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7956   {
7957     \@@_block_tikz:nnnnn
7958     { \seq_use:Nn \l_@@_tikz_seq { , } }
7959     { #1 }
7960     { #2 }
7961     { \int_use:N \l_@@_last_row_int }
7962     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of TikZ keys.

```

7962   }
7963 }
7964 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7965 {
7966   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7967   {
7968     \@@_actually_diagbox:nnnnnn
7969     { #1 }
7970     { #2 }
7971     { \int_use:N \l_@@_last_row_int }
7972     { \int_use:N \l_@@_last_col_int }
7973     { \exp_not:n { ##1 } }
7974     { \exp_not:n { ##2 } }

```

```

7975     }
7976 }

```

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.

```

7977 \pgfpicture
7978 \pgfrememberpicturepositiononpagetrue
7979 \pgf@relevantforpicturesizefalse
7980 \@@_qpoint:n { row - #1 }
7981 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7982 \@@_qpoint:n { col - #2 }
7983 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7984 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7985 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7986 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7987 \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.

```

7988 \@@_pgf_rect_node:nnnnn
7989 { \@@_env: - #1 - #2 - block }
7990 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7991 \str_if_empty:NF \l_@@_block_name_str
7992 {
7993   \pgfnodealias
7994   { \@@_env: - \l_@@_block_name_str }
7995   { \@@_env: - #1 - #2 - block }
7996   \str_if_empty:NF \l_@@_name_str
7997   {
7998     \pgfnodealias
7999     { \l_@@_name_str - \l_@@_block_name_str }
8000     { \@@_env: - #1 - #2 - block }
8001   }
8002 }

```

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.

```

8003 \bool_if:NF \l_@@_hpos_of_block_cap_bool
8004 {
8005   \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.

```

8006     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8007     {

```

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.

```

8008         \cs_if_exist:cT
8009         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8010         {
8011             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8012             {
8013                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
8014                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
8015             }
8016         }
8017     }

```

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.

```

8018     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
8019     {
8020         \@@_qpoint:n { col - #2 }
8021         \dim_set_eq:NN \l_tmpb_dim \pgf@x
8022     }
8023     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
8024     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8025     {
8026         \cs_if_exist:cT
8027         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8028         {
8029             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
8030             {
8031                 \pgfpointanchor
8032                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
8033                 { east }
8034                 \dim_set:Nn \l_@@_tmpd_dim
8035                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
8036             }
8037         }
8038     }
8039     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
8040     {
8041         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8042         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8043     }
8044     \@@_pgf_rect_node:nnnn
8045     { \@@_env: - #1 - #2 - block - short }
8046     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
8047 }

```

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.

```

8048     \bool_if:NT \l_@@_medium_nodes_bool
8049     {
8050         \@@_pgf_rect_node:nnn
8051         { \@@_env: - #1 - #2 - block - medium }
8052         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
8053         {
8054             \pgfpointanchor
8055             { \@@_env:
8056               - \int_use:N \l_@@_last_row_int
8057               - \int_use:N \l_@@_last_col_int - medium
8058             }

```

```

8059         { south-east }
8060     }
8061 }
8062 \endpgfpicture
8063

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contains* an ampersand &.

```

8064 \bool_if:NTF \l_@@_ampersand_bool
8065 {
8066     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
8067     \int_zero_new:N \l_@@_split_int
8068     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }

```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell.

```

8069     \int_zero_new:N \l_@@_first_row_int
8070     \int_zero_new:N \l_@@_first_col_int
8071     \int_zero_new:N \l_@@_last_row_int
8072     \int_zero_new:N \l_@@_last_col_int
8073     \int_set:Nn \l_@@_first_row_int { #1 }
8074     \int_set:Nn \l_@@_first_col_int { #2 }
8075     \int_set:Nn \l_@@_last_row_int { #3 }
8076     \int_set:Nn \l_@@_last_col_int { #4 }

8077     \pgfpicture
8078     \pgfrememberpicturerepositiononpagetrue
8079     \pgf@relevantforpicturesizefalse
8080     \@@_qpoint:n { row - #1 }
8081     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8082     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8083     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
8084     \@@_qpoint:n { col - #2 }
8085     \dim_set_eq:NN \l_tmpa_dim \pgf@x
8086     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8087     \dim_set:Nn \l_tmpb_dim
8088     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
8089     \bool_lazy_or:nnT
8090     { \l_@@_vlines_block_bool }
8091     { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
8092     {
8093         \int_step_inline:nn { \l_@@_split_int - 1 }
8094         {
8095             \pgfpathmoveto
8096             {
8097                 \pgfpoint
8098                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8099                 \l_@@_tmpc_dim
8100             }
8101             \pgfpathlineto
8102             {
8103                 \pgfpoint
8104                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
8105                 \l_@@_tmpd_dim
8106             }
8107             \CT@arc@
8108             \pgfsetlinewidth { 1.1 \arrayrulewidth }
8109             \pgfsetrectcap
8110             \pgfusepathqstroke
8111         }
8112     }
8113     \cs_set_eq:NN \cellcolor \@@_subcellcolor
8114     \int_zero_new:N \l_@@_split_i_int
8115     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }

```

```

8116 {
8117   \pgfpointanchor { \@@_env: - #1 - #2 - block } { north }
8118   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8119 }
8120 {
8121   \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8122   {
8123     \pgfpointanchor { \@@_env: - #1 - #2 - block } { south }
8124     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8125   }
8126   {
8127     \bool_lazy_or:nnTF
8128     { \int_compare_p:nNn { #1 } = { #3 } }
8129     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8130     {
8131       \@@_qpoint:n { row - #1 - base }
8132       \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8133     }
8134     {
8135       \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
8136       {
8137         \@@_qpoint:n { row - #3 - base }
8138         \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
8139       }
8140       {
8141         \@@_qpoint:n { #1 - #2 - block }
8142         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8143       }
8144     }
8145   }
8146 }
8147 \int_step_inline:nn { \l_@@_split_int }
8148 {
8149   \group_begin:

```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```

8150   \int_set:Nn \l_@@_split_i_int { ##1 }
8151   \dim_set:Nn \col@sep
8152   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
8153   \pgftransformshift
8154   {
8155     \pgfpoint
8156     {
8157       \l_tmpa_dim + ##1 \l_tmpb_dim -
8158       \str_case:on \l_@@_hpos_block_str
8159       {
8160         l { \l_tmpb_dim + \col@sep }
8161         c { 0.5 \l_tmpb_dim }
8162         r { \col@sep }
8163       }
8164     }
8165     { \l_@@_tmpc_dim }
8166   }
8167   \pgfset { inner~sep = \c_zero_dim }
8168   \pgfnode
8169   { rectangle }
8170   {
8171     \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
8172     {
8173       \str_case:on \l_@@_hpos_block_str
8174       {
8175         l { north-west }
8176         c { north }
8177         r { north-east }

```

```

8178     }
8179   }
8180   {
8181     \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
8182     {
8183       \str_case:on \l_@@_hpos_block_str
8184       {
8185         l { south~west }
8186         c { south }
8187         r { south~east }
8188       }
8189     }
8190     {
8191       \bool_lazy_any:nTF
8192       {
8193         { \int_compare_p:nNn { #1 } = { #3 } }
8194         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
8195         { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
8196       }
8197       {
8198         \str_case:on \l_@@_hpos_block_str
8199         {
8200           l { base~west }
8201           c { base }
8202           r { base~east }
8203         }
8204       }
8205       {
8206         \str_case:on \l_@@_hpos_block_str
8207         {
8208           l { west }
8209           c { center }
8210           r { east }
8211         }
8212       }
8213     }
8214   }
8215 }
8216 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
8217 \group_end:
8218 }
8219 \endpgfpicture
8220 }

```

Now the case where there is no ampersand & in the content of the block.

```

8221 {
8222   \bool_if:NTF \l_@@_p_block_bool
8223   {

```

When the final user has used the key p, we have to compute the width.

```

8224 \pgfpicture
8225 \pgfrememberpicturepositiononpagetrue
8226 \pgf@relevantforpicturesizefalse
8227 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
8228 {
8229   \@@_qpoint:n { col - #2 }
8230   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8231   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
8232 }
8233 {
8234   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
8235   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
8236   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
8237 }

```

```

8238         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
8239     \endpgfpicture
8240     \hbox_set:Nn \l_@@_cell_box
8241     {
8242         \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
8243             { \g_tmpb_dim }
8244             \str_case:on \l_@@_hpos_block_str
8245                 { c \centering r \raggedleft l \raggedright j { } }
8246             #6
8247         \end { minipage }
8248     }
8249 }
8250 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8251 \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.

```

8252 \pgfpicture
8253 \pgfrememberpicturepositiononpagetrue
8254 \pgf@relevantforpicturesizefalse
8255 \bool_lazy_any:nTF
8256 {
8257     { \str_if_empty_p:N \l_@@_vpos_block_str }
8258     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
8259     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
8260     { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
8261 }
8262 {

```

If we are in the “first column”, we must put the block as if it was with the key `r`.

```

8263     \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`.

```

8264     \bool_if:nT \g_@@_last_col_found_bool
8265     {
8266         \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
8267         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8268     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8269     \tl_set:Ne \l_tmpa_tl
8270     {
8271         \str_case:on \l_@@_vpos_block_str
8272         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8273         { } {
8274             \str_case:on \l_@@_hpos_block_str
8275             {
8276                 c { center }
8277                 l { west }
8278                 r { east }
8279                 j { center }
8280             }
8281         }
8282         c {
8283             \str_case:on \l_@@_hpos_block_str
8284             {
8285                 c { center }
8286                 l { west }
8287                 r { east }
8288                 j { center }

```



```

8289         }
8290
8291     }
8292     T {
8293         \str_case:on \l_@@_hpos_block_str
8294         {
8295             c { north }
8296             l { north~west }
8297             r { north~east }
8298             j { north }
8299         }
8300
8301     }
8302     B {
8303         \str_case:on \l_@@_hpos_block_str
8304         {
8305             c { south }
8306             l { south~west }
8307             r { south~east }
8308             j { south }
8309         }
8310
8311     }
8312 }
8313 }
8314 \pgftransformshift
8315 {
8316     \pgfpointanchor
8317     {
8318         \@@_env: - #1 - #2 - block
8319         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8320     }
8321     { \l_tmpa_tl }
8322 }
8323 \pgfset { inner~sep = \c_zero_dim }
8324 \pgfnode
8325 { rectangle }
8326 { \l_tmpa_tl }
8327 { \box_use_drop:N \l_@@_cell_box } { } { }
8328 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8329 {
8330     \pgfextracty \l_tmpa_dim
8331     {
8332         \@@_qpoint:n
8333         {
8334             row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
8335             - base
8336         }
8337     }
8338     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

8339 \pgfpointanchor
8340 {
8341     \@@_env: - #1 - #2 - block
8342     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8343 }
8344 {
8345     \str_case:on \l_@@_hpos_block_str
8346     {
8347         c { center }

```

```

8348         l { west }
8349         r { east }
8350         j { center }
8351     }
8352 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8353     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8354     \pgfset { inner~sep = \c_zero_dim }
8355     \pgfnode
8356     { rectangle }
8357     {
8358         \str_case:on \l_@@_hpos_block_str
8359         {
8360             c { base }
8361             l { base~west }
8362             r { base~east }
8363             j { base }
8364         }
8365     }
8366     { \box_use_drop:N \l_@@_cell_box } { } { }
8367 }
8368 \endpgfpicture
8369 }
8370 \group_end:
8371 }
8372 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8373 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8374 {
8375     \tl_if_empty:NF \l_@@_opacity_tl
8376     {
8377         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8378             {
8379                 \tl_set:Nc \l_@@_fill_tl
8380                 {
8381                     [ opacity = \l_@@_opacity_tl ,
8382                     \tl_tail:o \l_@@_fill_tl
8383                 }
8384             }
8385             {
8386                 \tl_set:Nc \l_@@_fill_tl
8387                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8388             }
8389         }
8390     }

```

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).

```

8391 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8392 {
8393     \group_begin:
8394     \tl_clear:N \l_@@_draw_tl
8395     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8396     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8397     \pgfpicture
8398     \pgfrememberpicturepositiononpagetrue
8399     \pgf@relevantforpicturesizefalse

```

```

8400 \tl_if_empty:NF \l_@@_draw_tl
8401 {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8402 \tl_if_eq:NnTF \l_@@_draw_tl { default }
8403 { \CT@arc@ }
8404 { \@@_color:o \l_@@_draw_tl }
8405 }
8406 \pgfsetcornersarced
8407 {
8408 \pgfpoint
8409 { \l_@@_rounded_corners_dim }
8410 { \l_@@_rounded_corners_dim }
8411 }
8412 \@@_cut_on_hyphen:w #2 \q_stop
8413 \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8414 {
8415 \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8416 {
8417 \@@_qpoint:n { row - \l_tmpa_tl }
8418 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8419 \@@_qpoint:n { col - \l_tmpb_tl }
8420 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8421 \@@_cut_on_hyphen:w #3 \q_stop
8422 \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8423 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8424 \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8425 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8426 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8427 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8428 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8429 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8430 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8431 \pgfpathrectanglecorners
8432 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8433 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8434 \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8435 { \pgfusepathqstroke }
8436 { \pgfusepath { stroke } }
8437 }
8438 }
8439 \endpgfpicture
8440 \group_end:
8441 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8442 \keys_define:nn { nicematrix / BlockStroke }
8443 {
8444 color .tl_set:N = \l_@@_draw_tl ,
8445 draw .code:n =
8446 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8447 draw .default:n = default ,
8448 line-width .dim_set:N = \l_@@_line_width_dim ,
8449 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8450 rounded-corners .default:n = 4 pt
8451 }

```

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).

```

8452 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8453 {

```

```

8454 \group_begin:
8455 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8456 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8457 \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8458 \@@_cut_on_hyphen:w #2 \q_stop
8459 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8460 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8461 \@@_cut_on_hyphen:w #3 \q_stop
8462 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8463 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8464 \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8465 {
8466   \use:e
8467   {
8468     \@@_vline:n
8469     {
8470       position = ##1 ,
8471       start = \l_@@_tmpc_tl ,
8472       end = \int_eval:n { \l_tmpa_tl - 1 } ,
8473       total-width = \dim_use:N \l_@@_line_width_dim
8474     }
8475   }
8476 }
8477 \group_end:
8478 }

8479 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8480 {
8481   \group_begin:
8482   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8483   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8484   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8485   \@@_cut_on_hyphen:w #2 \q_stop
8486   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8487   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8488   \@@_cut_on_hyphen:w #3 \q_stop
8489   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8490   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8491   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8492   {
8493     \use:e
8494     {
8495       \@@_hline:n
8496       {
8497         position = ##1 ,
8498         start = \l_@@_tmpd_tl ,
8499         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8500         total-width = \dim_use:N \l_@@_line_width_dim
8501       }
8502     }
8503   }
8504   \group_end:
8505 }

```

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).

```

8506 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8507 {
8508   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8509   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8510   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8511   { \@@_error:n { borders~forbidden } }

```

```

8512 {
8513   \tl_clear_new:N \l_@@_borders_tikz_tl
8514   \keys_set:no
8515     { nicematrix / OnlyForTikzInBorders }
8516   \l_@@_borders_clist
8517   \@@_cut_on_hyphen:w #2 \q_stop
8518   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8519   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8520   \@@_cut_on_hyphen:w #3 \q_stop
8521   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8522   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8523   \@@_stroke_borders_block_i:
8524 }
8525 }

8526 \hook_gput_code:nnn { begindocument } { . }
8527 {
8528   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8529   {
8530     \c_@@_pgfortikzpicture_tl
8531     \@@_stroke_borders_block_ii:
8532     \c_@@_endpgfortikzpicture_tl
8533   }
8534 }

8535 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8536 {
8537   \pgfrememberpicturepositiononpagetrue
8538   \pgf@relevantforpicturesizefalse
8539   \CT@arc@
8540   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8541   \clist_if_in:NnT \l_@@_borders_clist { right }
8542     { \@@_stroke_vertical:n \l_tmpb_tl }
8543   \clist_if_in:NnT \l_@@_borders_clist { left }
8544     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8545   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8546     { \@@_stroke_horizontal:n \l_tmpa_tl }
8547   \clist_if_in:NnT \l_@@_borders_clist { top }
8548     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8549 }

8550 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8551 {
8552   tikz .code:n =
8553     \cs_if_exist:NTF \tikzpicture
8554       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8555       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8556   tikz .value_required:n = true ,
8557   top .code:n = ,
8558   bottom .code:n = ,
8559   left .code:n = ,
8560   right .code:n = ,
8561   unknown .code:n = \@@_error:n { bad~border }
8562 }

```

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).

```

8563 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8564 {
8565   \@@_qpoint:n \l_@@_tmpc_tl
8566   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8567   \@@_qpoint:n \l_tmpa_tl
8568   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8569   \@@_qpoint:n { #1 }
8570   \tl_if_empty:NTF \l_@@_borders_tikz_tl

```

```

8571 {
8572   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8573   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8574   \pgfusepathqstroke
8575 }
8576 {
8577   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8578   ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8579 }
8580 }

```

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).

```

8581 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8582 {
8583   \@@_qpoint:n \l_@@_tmpd_tl
8584   \clist_if_in:NnTF \l_@@_borders_clist { left }
8585     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8586     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8587   \@@_qpoint:n \l_tmpb_tl
8588   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8589   \@@_qpoint:n { #1 }
8590   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8591     {
8592       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8593       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8594       \pgfusepathqstroke
8595     }
8596     {
8597       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8598       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8599     }
8600 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

8601 \keys_define:nn { nicematrix / BlockBorders }
8602 {
8603   borders .clist_set:N = \l_@@_borders_clist ,
8604   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8605   rounded-corners .default:n = 4 pt ,
8606   line-width .dim_set:N = \l_@@_line_width_dim
8607 }

```

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.

```

8608 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8609 {
8610   \begin { tikzpicture }
8611   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8612   \clist_map_inline:nn { #1 }
8613   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8614 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8615 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8616 (
8617 [
8618 xshift = \dim_use:N \l_@@_offset_dim ,
8619 yshift = - \dim_use:N \l_@@_offset_dim
8620 ]
8621 #2 -| #3
8622 )
8623 rectangle
8624 (
8625 [
8626 xshift = - \dim_use:N \l_@@_offset_dim ,
8627 yshift = \dim_use:N \l_@@_offset_dim
8628 ]
8629 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8630 ) ;
8631 }
8632 \end { tikzpicture }
8633 }
8634 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

8635 \keys_define:nn { nicematrix / SpecialOffset }
8636 { 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.

```

8637 \cs_new_protected:Npn \@@_NullBlock:
8638 { \@@_collect_options:n { \@@_NullBlock_i: } }
8639 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8640 { }

```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (&). Of course, `&-in-blocks` must be in force.

```

8641 \NewDocumentCommand \@@_subcellcolor { 0 { } m }
8642 {
8643 \tl_gput_right:Ne \g_@@_pre_code_before_tl
8644 {

```

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`).

```

8645 \@@_subcellcolor:nnnnnnn
8646 {
8647 \tl_if_blank:nTF { #1 }
8648 { { \exp_not:n { #2 } } }
8649 { [ #1 ] { \exp_not:n { #2 } } }
8650 }
8651 { \int_use:N \l_@@_first_row_int } % first row of the block
8652 { \int_use:N \l_@@_first_col_int } % first column of the block
8653 { \int_use:N \l_@@_last_row_int } % last row of the block
8654 { \int_use:N \l_@@_last_col_int } % last column of the block
8655 { \int_use:N \l_@@_split_int }
8656 { \int_use:N \l_@@_split_i_int }
8657 }
8658 \ignorespaces
8659 }

8660 \cs_new_protected:Npn \@@_subcellcolor:nnnnnn #1 #2 #3 #4 #5 #6 #7
8661 {
8662 \@@_color_opacity: #1

```

```

8663 \pgfpicture
8664 \pgf@relevantforpicturesizefalse
8665 \@@_qpoint:n { col - #3 }
8666 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8667 \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
8668 \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
8669 \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
8670 \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
8671 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8672 \@@_qpoint:n { row - #2 }
8673 \pgfpathrectanglecorners
8674 { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
8675 { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
8676 \pgfusepathqfill
8677 \endpgfpicture
8678 }

```

## 27 How to draw the dotted lines transparently

```

8679 \cs_set_protected:Npn \@@_renew_matrix:
8680 {
8681   \RenewDocumentEnvironment { pmatrix } { } {
8682     { \pNiceMatrix }
8683     { \endpNiceMatrix }
8684   \RenewDocumentEnvironment { vmatrix } { } {
8685     { \vNiceMatrix }
8686     { \endvNiceMatrix }
8687   \RenewDocumentEnvironment { Vmatrix } { } {
8688     { \VNiceMatrix }
8689     { \endVNiceMatrix }
8690   \RenewDocumentEnvironment { bmatrix } { } {
8691     { \bNiceMatrix }
8692     { \endbNiceMatrix }
8693   \RenewDocumentEnvironment { Bmatrix } { } {
8694     { \BNiceMatrix }
8695     { \endBNiceMatrix }
8696   }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8697 \keys_define:nn { nicematrix / Auto }
8698 {
8699   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8700   columns-type .value_required:n = true ,
8701   l .meta:n = { columns-type = l } ,
8702   r .meta:n = { columns-type = r } ,
8703   c .meta:n = { columns-type = c } ,
8704   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8705   delimiters / color .value_required:n = true ,
8706   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8707   delimiters / max-width .default:n = true ,
8708   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8709   delimiters .value_required:n = true ,
8710   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8711   rounded-corners .default:n = 4 pt
8712 }

```



```

8713 \NewDocumentCommand \AutoNiceMatrixWithDelims
8714 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8715 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8716 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8717 {

```

The group is for the protection of the keys.

```

8718 \group_begin:
8719 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8720 \use:e
8721 {
8722 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8723 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8724 [ \exp_not:o \l_tmpa_tl ]
8725 }
8726 \int_if_zero:nT { \l_@@_first_row_int }
8727 {
8728 \int_if_zero:nT { \l_@@_first_col_int } { & }
8729 \prg_replicate:nn { #4 - 1 } { & }
8730 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8731 }
8732 \prg_replicate:nn { #3 }
8733 {
8734 \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).

```

8735 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8736 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8737 }
8738 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8739 {
8740 \int_if_zero:nT { \l_@@_first_col_int } { & }
8741 \prg_replicate:nn { #4 - 1 } { & }
8742 \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\\
8743 }
8744 \end { NiceArrayWithDelims }
8745 \group_end:
8746 }

8747 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8748 {
8749 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8750 {
8751 \bool_gset_true:N \g_@@_delims_bool
8752 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8753 \AutoNiceMatrixWithDelims { #2 } { #3 }
8754 }
8755 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8756 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8757 {
8758 \group_begin:
8759 \bool_gset_false:N \g_@@_delims_bool
8760 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8761 \group_end:
8762 }

```

## 29 The redefinition of the command `\dotfill`

```

8763 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8764 \cs_new_protected:Npn \@@_dotfill:
8765 {

```

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}`).

```

8766 \@@_old_dotfill:
8767 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8768 }

```

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`.

```

8769 \cs_new_protected:Npn \@@_dotfill_i:
8770 {
8771   \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8772   { \@@_old_dotfill: }
8773 }

```

## 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.

```

8774 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8775 {
8776   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8777   {
8778     \@@_actually_diagbox:nnnnnn
8779     { \int_use:N \c@iRow }
8780     { \int_use:N \c@jCol }
8781     { \int_use:N \c@iRow }
8782     { \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.

```

8783   { \g_@@_row_style_tl \exp_not:n { #1 } }
8784   { \g_@@_row_style_tl \exp_not:n { #2 } }
8785 }

```

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`.

```

8786 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8787 {
8788   { \int_use:N \c@iRow }
8789   { \int_use:N \c@jCol }
8790   { \int_use:N \c@iRow }
8791   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8792   { }
8793 }
8794 }

```

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.

```

8795 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8796 {
8797   \pgfpicture
8798   \pgf@relevantforpicturesizefalse
8799   \pgfrememberpicturepositiononpagetrue
8800   \@@_qpoint:n { row - #1 }
8801   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8802   \@@_qpoint:n { col - #2 }
8803   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8804   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8805   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8806   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8807   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8808   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8809   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8810   {

```

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.

```

8811   \CT@arc@
8812   \pgfsetroundcap
8813   \pgfusepathqstroke
8814 }
8815 \pgfset { inner~sep = 1 pt }
8816 \pgfscope
8817 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8818 \pgfnode { rectangle } { south~west }
8819 {
8820   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8821   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8822   \end { minipage }
8823 }
8824 { }
8825 { }
8826 \endpgfscope
8827 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8828 \pgfnode { rectangle } { north~east }
8829 {
8830   \begin { minipage } { 20 cm }
8831   \raggedleft
8832   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8833   \end { minipage }
8834 }
8835 { }
8836 { }
8837 \endpgfpicture
8838 }

```

## 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. 87.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8839 \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 `\`.

```
8840 \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`.

```
8841 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8842 {
8843   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8844   \@@_CodeAfter_iv:n
8845 }
```

We catch the argument of the command `\end` (in `#1`).

```
8846 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8847 {
```

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.

```
8848   \str_if_eq:eeTF { \currentenv } { #1 }
8849   { \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`.

```
8850   {
8851     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8852     \@@_CodeAfter_ii:n
8853   }
8854 }
```

## 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.

```
8855 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8856 {
8857   \pgfpicture
8858   \pgfrememberpicturepositiononpagetrue
8859   \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.

```
8860   \@@_qpoint:n { row - 1 }
8861   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8862   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8863   \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).

```

8864 \bool_if:nTF { #3 }
8865 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8866 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8867 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8868 {
8869   \cs_if_exist:cT
8870   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8871   {
8872     \pgfpointanchor
8873     { \@@_env: - ##1 - #2 }
8874     { \bool_if:nTF { #3 } { west } { east } }
8875     \dim_set:Nn \l_tmpa_dim
8876     {
8877       \bool_if:nTF { #3 }
8878       { \dim_min:nn }
8879       { \dim_max:nn }
8880       \l_tmpa_dim
8881       { \pgf@x }
8882     }
8883   }
8884 }

```

Now we can put the delimiter with a node of PGF.

```

8885 \pgfset { inner~sep = \c_zero_dim }
8886 \dim_zero:N \nulldelimiterspace
8887 \pgftransformshift
8888 {
8889   \pgfpoint
8890   { \l_tmpa_dim }
8891   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8892 }
8893 \pgfnode
8894 { rectangle }
8895 { \bool_if:nTF { #3 } { east } { west } }
8896 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8897 \nullfont
8898 $ % $
8899 \@@_color:o \l_@@_delimiters_color_tl
8900 \bool_if:nTF { #3 } { \left #1 } { \left . }
8901 \vcenter
8902 {
8903   \nullfont
8904   \hrule \@height
8905   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8906   \@depth \c_zero_dim
8907   \@width \c_zero_dim
8908 }
8909 \bool_if:nTF { #3 } { \right . } { \right #1 }
8910 $ % $
8911 }
8912 { }
8913 { }
8914 \endpgfpicture
8915 }

```

### 33 The command `\SubMatrix`

```

8916 \keys_define:nn { nicematrix / sub-matrix }
8917 {
8918   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8919   extra-height .value_required:n = true ,
8920   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8921   left-xshift .value_required:n = true ,
8922   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8923   right-xshift .value_required:n = true ,
8924   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8925   xshift .value_required:n = true ,
8926   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8927   delimiters / color .value_required:n = true ,
8928   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8929   slim .default:n = true ,
8930   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8931   hlines .default:n = all ,
8932   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8933   vlines .default:n = all ,
8934   hvlines .meta:n = { hlines, vlines } ,
8935   hvlines .value_forbidden:n = true
8936 }
8937 \keys_define:nn { nicematrix }
8938 {
8939   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8940   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8941   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8942   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8943 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8944 \keys_define:nn { nicematrix / SubMatrix }
8945 {
8946   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8947   delimiters / color .value_required:n = true ,
8948   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8949   hlines .default:n = all ,
8950   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8951   vlines .default:n = all ,
8952   hvlines .meta:n = { hlines, vlines } ,
8953   hvlines .value_forbidden:n = true ,
8954   name .code:n =
8955     \tl_if_empty:nTF { #1 }
8956     { \@@_error:n { Invalid-name } }
8957     {
8958       \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8959       {
8960         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8961         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8962         {
8963           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8964           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8965         }
8966       }
8967       { \@@_error:n { Invalid-name } }
8968     } ,
8969   name .value_required:n = true ,
8970   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8971   rules .value_required:n = true ,
8972   code .tl_set:N = \l_@@_code_tl ,
8973   code .value_required:n = true ,
8974   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8975 }

```

```

8976 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8977 {
8978   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8979   {
8980     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8981     [
8982       delimiters / color = \l_@@_delimiters_color_tl ,
8983       hlines = \l_@@_submatrix_hlines_clist ,
8984       vlines = \l_@@_submatrix_vlines_clist ,
8985       extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8986       left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8987       right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8988       slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8989       #5
8990     ]
8991   }
8992   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8993   \ignorespaces
8994 }
8995 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8996 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8997 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8998 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8999 {
9000   \seq_gput_right:Ne \g_@@_submatrix_seq
9001   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

9002   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
9003   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
9004   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
9005   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
9006   }
9007 }

```

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).

```

9008 \NewDocumentCommand \@@_compute_i_j:nn
9009 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
9010 { \@@_compute_i_j:nnnn #1 #2 }
9011 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
9012 {
9013   \def \l_@@_first_i_tl { #1 }
9014   \def \l_@@_first_j_tl { #2 }
9015   \def \l_@@_last_i_tl { #3 }
9016   \def \l_@@_last_j_tl { #4 }
9017   \tl_if_eq:NnT \l_@@_first_i_tl { last }
9018     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
9019   \tl_if_eq:NnT \l_@@_first_j_tl { last }
9020     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
9021   \tl_if_eq:NnT \l_@@_last_i_tl { last }
9022     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
9023   \tl_if_eq:NnT \l_@@_last_j_tl { last }
9024     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
9025 }

```

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`.

```

9026 \hook_gput_code:nnn { begindocument } { . }
9027 {
9028   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
9029   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
9030     { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
9031 }
9032 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
9033 {
9034   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

9035   \@@_compute_i_j:nn { #2 } { #3 }
9036   \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
9037   { \def \arraystretch { 1 } }
9038   \bool_lazy_or:nnTF
9039     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9040     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9041     { \@@_error:nn { Construct~too-large } { \SubMatrix } }
9042   {
9043     \str_clear_new:N \l_@@_submatrix_name_str
9044     \keys_set:nn { nicematrix / SubMatrix } { #5 }
9045     \pgfpicture
9046     \pgfrememberpicturepositiononpagetrue
9047     \pgf@relevantforpicturesizefalse
9048     \pgfset { inner~sep = \c_zero_dim }
9049     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9050     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curryfication.

```

9051   \bool_if:NTF \l_@@_submatrix_slim_bool
9052   { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
9053   { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
9054   {
9055     \cs_if_exist:cT
9056     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9057     {
9058       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9059       \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9060       { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9061     }
9062     \cs_if_exist:cT
9063     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9064     {
9065       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9066       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9067       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9068     }
9069   }
9070   \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
9071   { \@@_error:nn { Impossible~delimiter } { left } }
9072   {
9073     \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }

```



```

9074         { \@@_error:nn { Impossible-delimiter } { right } }
9075         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
9076     }
9077     \endpgfpicture
9078 }
9079 \group_end:
9080 \ignorespaces
9081 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

9082 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
9083 {
9084     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
9085     \dim_set:Nn \l_@@_y_initial_dim
9086     {
9087         \fp_to_dim:n
9088         {
9089             \pgf@y
9090             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
9091         }
9092     }
9093     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
9094     \dim_set:Nn \l_@@_y_final_dim
9095     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
9096     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
9097     {
9098         \cs_if_exist:cT
9099         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
9100         {
9101             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
9102             \dim_set:Nn \l_@@_y_initial_dim
9103             { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
9104         }
9105         \cs_if_exist:cT
9106         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
9107         {
9108             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
9109             \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
9110             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
9111         }
9112     }
9113     \dim_set:Nn \l_tmpa_dim
9114     {
9115         \l_@@_y_initial_dim - \l_@@_y_final_dim +
9116         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
9117     }
9118     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

9119     \group_begin:
9120     \pgfsetlinewidth { 1.1 \arrayrulewidth }
9121     \@@_set_CTarc:o \l_@@_rules_color_tl
9122     \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`.

```

9123     \seq_map_inline:Nn \g_@@_cols_vlism_seq
9124     {
9125         \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
9126         {
9127             \int_compare:nNnT
9128             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }

```

```
9129         {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
9130             \@_qpoint:n { col - ##1 }
9131             \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9132             \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9133             \pgfusepathqstroke
9134         }
9135     }
9136 }
```

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.

```
9137 \str_if_eq:eeTF { \l_@_submatrix_vlines_clist } { all }
9138 { \int_step_inline:nn { \l_@_last_j_tl - \l_@_first_j_tl } }
9139 { \clist_map_inline:Nn \l_@_submatrix_vlines_clist }
9140 {
9141     \bool_lazy_and:nnTF
9142     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9143     {
9144         \int_compare_p:nNn
9145         { ##1 } < { \l_@_last_j_tl - \l_@_first_j_tl + 1 } }
9146     {
9147         \@_qpoint:n { col - \int_eval:n { ##1 + \l_@_first_j_tl } }
9148         \pgfpathmoveto { \pgfpoint \pgf@x \l_@_y_initial_dim }
9149         \pgfpathlineto { \pgfpoint \pgf@x \l_@_y_final_dim }
9150         \pgfusepathqstroke
9151     }
9152     { \@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
9153 }
```

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.

```
9154 \str_if_eq:eeTF { \l_@_submatrix_hlines_clist } { all }
9155 { \int_step_inline:nn { \l_@_last_i_tl - \l_@_first_i_tl } }
9156 { \clist_map_inline:Nn \l_@_submatrix_hlines_clist }
9157 {
9158     \bool_lazy_and:nnTF
9159     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
9160     {
9161         \int_compare_p:nNn
9162         { ##1 } < { \l_@_last_i_tl - \l_@_first_i_tl + 1 } }
9163     {
9164         \@_qpoint:n { row - \int_eval:n { ##1 + \l_@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
9165     \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
9166     \dim_set:Nn \l_tmpa_dim
9167     { \l_@_x_initial_dim - \l_@_submatrix_left_xshift_dim }
9168     \str_case:nn { #1 }
9169     {
9170         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9171         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
9172         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
9173     }
9174     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```
9175     \dim_set:Nn \l_tmpb_dim
9176     { \l_@_x_final_dim + \l_@_submatrix_right_xshift_dim }
9177     \str_case:nn { #2 }
9178     {
9179         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
```

```

9180         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
9181         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
9182     }
9183     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
9184     \pgfusepathqstroke
9185     \group_end:
9186 }
9187 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
9188 }

```

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).

```

9189 \str_if_empty:NF \l_@@_submatrix_name_str
9190 {
9191     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
9192     \l_@@_x_initial_dim \l_@@_y_initial_dim
9193     \l_@@_x_final_dim \l_@@_y_final_dim
9194 }
9195 \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`.

```

9196 \begin { pgfscope }
9197 \pgftransformshift
9198 {
9199     \pgfpoint
9200     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
9201     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9202 }
9203 \str_if_empty:NTF \l_@@_submatrix_name_str
9204 { \@@_node_left:nn #1 { } }
9205 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
9206 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

9207 \pgftransformshift
9208 {
9209     \pgfpoint
9210     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
9211     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
9212 }
9213 \str_if_empty:NTF \l_@@_submatrix_name_str
9214 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
9215 {
9216     \@@_node_right:nnnn #2
9217     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
9218 }

```

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`.

```

9219 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
9220 \flag_clear_new:N \l_@@_code_flag
9221 \l_@@_code_tl
9222 }

```

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`.

```

9223 \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.

```
9224 \cs_new:Npn \@@_pgfpointanchor:n #1
9225 { \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`).

```
9226 \cs_new:Npn \@@_pgfpointanchor_i:n #1
9227 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

9228 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
9229 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9230 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
9231 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
9232 { \@@_pgfpointanchor_ii:n { #1 } }
9233 }
```

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`).

```
9234 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
9235 { \@@_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`.

```
9236 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }

9237 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
9238 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
9239 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
9240 { \@@_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`).

```
9241 { \@@_pgfpointanchor_iii:w { #1 } #2 }
9242 }
```

The following function is for the case when the name contains an hyphen.

```
9243 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
9244 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
9245 \@@_env:
9246 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
9247 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
9248 }
```

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.

```

9249 \tl_const:Nn \c_@@_integers_alist_tl
9250 {
9251   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
9252   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
9253   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
9254   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
9255 }

9256 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
9257 {

```

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`.

```

9258   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
9259   {
9260     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

9261   \@@_env: -
9262   \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9263   { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
9264   { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
9265 }
9266 {
9267   \str_if_eq:eeTF { #1 } { last }
9268   {
9269     \flag_raise:N \l_@@_code_flag
9270     \@@_env: -
9271     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9272     { \int_eval:n { \l_@@_last_i_tl + 1 } }
9273     { \int_eval:n { \l_@@_last_j_tl + 1 } }
9274   }
9275   { #1 }
9276 }
9277 }

```

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`).

```

9278 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9279 {
9280   \pgfnode
9281   { rectangle }
9282   { east }
9283   {
9284     \nullfont
9285     $ % $
9286     \@@_color:o \l_@@_delimiters_color_tl
9287     \left #1
9288     \vcenter
9289     {
9290       \nullfont
9291       \hrule \@height \l_tmpa_dim
9292       \@depth \c_zero_dim

```

```

9293             \@width \c_zero_dim
9294         }
9295         \right .
9296         $ % $
9297     }
9298     { #2 }
9299     { }
9300 }

```

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.

```

9301 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9302 {
9303     \pgfnode
9304     { rectangle }
9305     { west }
9306     {
9307         \nullfont
9308         $ % $
9309         \colorlet { current-color } { . }
9310         \@@_color:o \l_@@_delimiters_color_tl
9311         \left .
9312         \vcenter
9313         {
9314             \nullfont
9315             \hrule \@height \l_tmpa_dim
9316                 \@depth \c_zero_dim
9317                 \@width \c_zero_dim
9318         }
9319         \right #1
9320         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9321         ^ { \color { current-color } \smash { #4 } }
9322         $ % $
9323     }
9324     { #2 }
9325     { }
9326 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9327 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9328 {
9329     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9330     \ignorespaces
9331 }
9332 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9333 {
9334     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9335     \ignorespaces
9336 }
9337 \keys_define:nn { nicematrix / Brace }
9338 {
9339     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9340     left-shorten .default:n = true ,
9341     left-shorten .value_forbidden:n = true ,

```

```

9342   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9343   right-shorten .default:n = true ,
9344   right-shorten .value_forbidden:n = true ,
9345   shorten .meta:n = { left-shorten , right-shorten } ,
9346   shorten .value_forbidden:n = true ,
9347   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9348   yshift .value_required:n = true ,
9349   yshift .initial:n = \c_zero_dim ,
9350   color .tl_set:N = \l_tmpa_tl ,
9351   color .value_required:n = true ,
9352   unknown .code:n =
9353     \@@_unknown_key:nn
9354     { nicematrix / Brace }
9355     { Unknown-key-for-Brace }
9356 }

```

#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*.

```

9357 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9358 {
9359   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9360   \@@_compute_i_j:nn { #1 } { #2 }
9361   \bool_lazy_or:nnTF
9362     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9363     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9364     {
9365       \str_if_eq:eeTF { #5 } { under }
9366       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9367       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9368     }
9369     {
9370       \tl_clear:N \l_tmpa_tl
9371       \keys_set:nn { nicematrix / Brace } { #4 }
9372       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9373       \pgfpicture
9374       \pgfrememberpicturepositiononpagetrue
9375       \pgf@relevantforpicturesizefalse
9376       \bool_if:NT \l_@@_brace_left_shorten_bool
9377       {
9378         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9379         \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9380         {
9381           \cs_if_exist:cT
9382             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9383             {
9384               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9385
9386               \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9387               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9388             }
9389         }
9390       }
9391       \bool_lazy_or:nnT
9392         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9393         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9394         {
9395           \@@_qpoint:n { col - \l_@@_first_j_tl }
9396           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9397         }
9398       \bool_if:NT \l_@@_brace_right_shorten_bool
9399       {

```

```

9400 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9401 \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9402 {
9403   \cs_if_exist:cT
9404   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9405   {
9406     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9407     \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9408     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9409   }
9410 }
9411 }
9412 \bool_lazy_or:nnT
9413 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9414 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9415 {
9416   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9417   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9418 }
9419 \pgfset { inner~sep = \c_zero_dim }
9420 \str_if_eq:eeTF { #5 } { under }
9421 { \@@_underbrace_i:n { #3 } }
9422 { \@@_overbrace_i:n { #3 } }
9423 \endpgfpicture
9424 }
9425 \group_end:
9426 }

```

The argument is the text to put above the brace.

```

9427 \cs_new_protected:Npn \@@_overbrace_i:n #1
9428 {
9429   \@@_qpoint:n { row - \l_@@_first_i_tl }
9430   \pgftransformshift
9431   {
9432     \pgfpoint
9433     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9434     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9435   }
9436   \pgfnode
9437   { rectangle }
9438   { south }
9439   {
9440     \vtop
9441     {
9442       \group_begin:
9443       \everycr { }
9444       \halign
9445       {
9446         \hfil ## \hfil \crcr
9447         \bool_if:NTF \l_@@_tabular_bool
9448         { \begin { tabular } { c } #1 \end { tabular } }
9449         { $ \begin { array } { c } #1 \end { array } $ }
9450         \cr
9451         $ % $
9452         \overbrace
9453         {
9454           \hbox_to_wd:nn
9455           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9456           { }
9457         }
9458         $ % $
9459         \cr
9460       }
9461       \group_end:

```



```

9462     }
9463   }
9464   { }
9465   { }
9466 }

```

The argument is the text to put under the brace.

```

9467 \cs_new_protected:Npn \l_@@_underbrace_i:n #1
9468 {
9469   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9470   \pgftransformshift
9471   {
9472     \pgfpoint
9473     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9474     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9475   }
9476   \pgfnode
9477   { rectangle }
9478   { north }
9479   {
9480     \group_begin:
9481     \everycr { }
9482     \vbox
9483     {
9484       \halign
9485       {
9486         \hfil ## \hfil \crcr
9487         $ % $
9488         \underbrace
9489         {
9490           \hbox_to_wd:nn
9491           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9492           { }
9493         }
9494         $ % $
9495         \cr
9496         \bool_if:NTF \l_@@_tabular_bool
9497         { \begin { tabular } { c } #1 \end { tabular } }
9498         { $ \begin { array } { c } #1 \end { array } $ }
9499         \cr
9500       }
9501     }
9502     \group_end:
9503   }
9504   { }
9505   { }
9506 }

```

## 35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.

We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```

9507 \AddToHook { package / tikz / after }
9508 {

```

```

9509 \tikzset
9510 {
9511     nicematrix / brace / .style =
9512     {
9513         decoration = { brace , raise = -0.15 em } ,
9514         decorate ,
9515     } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9516     nicematrix / mirrored-brace / .style =
9517     {
9518         nicematrix / brace ,
9519         decoration = mirror ,
9520     }
9521 }
9522 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9523 \keys_define:nn { nicematrix / Hbrace }
9524 {
9525     color .code:n = ,
9526     horizontal-label .code:n = ,
9527     horizontal-labels .code:n = ,
9528     shorten .code:n = ,
9529     shorten-start .code:n = ,
9530     shorten-end .code:n = ,
9531     shorten+ .code:n = ,
9532     shorten-start+ .code:n = ,
9533     shorten-end+ .code:n = ,
9534     shorten~+ .code:n = ,
9535     shorten-start~+ .code:n = ,
9536     shorten-end~+ .code:n = ,
9537     brace-shift .code:n = ,
9538     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9539 }

```

Here we need an “fully expandable” command.

```

9540 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9541 {
9542     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9543     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9544     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9545 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9546 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9547 {
9548     \int_compare:nNnTF { \c@iRow } < { 2 }
9549     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9550     \str_if_eq:nnTF { #2 } { * }
9551     {
9552         \bool_set_true:N \l_@@_nullify_dots_bool
9553         \Ldots
9554         [
9555             line-style = nicematrix / brace ,
9556             #1 ,
9557             up =
9558             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }

```

```

9559     ]
9560   }
9561   {
9562     \Hdotsfor
9563     [
9564       line-style = nicematrix / brace ,
9565       #1 ,
9566       up =
9567         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9568     ]
9569     { #2 }
9570   }
9571 }
9572 {
9573   \str_if_eq:nnTF { #2 } { * }
9574   {
9575     \bool_set_true:N \l_@@_nullify_dots_bool
9576     \Ldots
9577     [
9578       line-style = nicematrix / mirrored-brace ,
9579       #1 ,
9580       down =
9581         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9582     ]
9583   }
9584   {
9585     \Hdotsfor
9586     [
9587       line-style = nicematrix / mirrored-brace ,
9588       #1 ,
9589       down =
9590         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9591     ]
9592     { #2 }
9593   }
9594 }
9595 \keys_set:nn { nicematrix / Hbrace } { #1 }
9596 }

9597 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9598 {
9599   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9600   { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9601   { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9602 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9603 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9604 {
9605   \int_compare:nNnTF { \c@jCol } < { 2 }
9606   {
9607     \str_if_eq:nnTF { #2 } { * }
9608     {
9609       \bool_set_true:N \l_@@_nullify_dots_bool
9610       \Vdots
9611       [
9612         Vbrace ,
9613         line-style = nicematrix / mirrored-brace ,
9614         #1 ,
9615         down =
9616           \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9617       ]

```

```

9618     }
9619     {
9620         \Vdotsfor
9621         [
9622             Vbrace ,
9623             line-style = nicematrix / mirrored-brace ,
9624             #1 ,
9625             down =
9626                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9627         ]
9628         { #2 }
9629     }
9630 }
9631 {
9632     \str_if_eq:nnTF { #2 } { * }
9633     {
9634         \bool_set_true:N \l_@@_nullify_dots_bool
9635         \Vdots
9636         [
9637             Vbrace ,
9638             line-style = nicematrix / brace ,
9639             #1 ,
9640             up =
9641                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9642         ]
9643     }
9644     {
9645         \Vdotsfor
9646         [
9647             Vbrace ,
9648             line-style = nicematrix / brace ,
9649             #1 ,
9650             up =
9651                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9652         ]
9653         { #2 }
9654     }
9655 }
9656 \keys_set:nn { nicematrix / Hbrace } { #1 }
9657 }

```

## 36 The command TikzEveryCell

```

9658 \bool_new:N \l_@@_not_empty_bool
9659 \bool_new:N \l_@@_empty_bool
9660
9661 \keys_define:nn { nicematrix / TikzEveryCell }
9662 {
9663     not-empty .code:n =
9664         \bool_lazy_or:nnTF
9665         { \l_@@_in_code_after_bool }
9666         { \g_@@_create_cell_nodes_bool }
9667         { \bool_set_true:N \l_@@_not_empty_bool }
9668         { \@@_error:n { detection~of~empty~cells } } ,
9669     not-empty .value_forbidden:n = true ,
9670     empty .code:n =
9671         \bool_lazy_or:nnTF
9672         { \l_@@_in_code_after_bool }
9673         { \g_@@_create_cell_nodes_bool }

```

```

9674     { \bool_set_true:N \l_@@_empty_bool }
9675     { \@@_error:n { detection-of-empty-cells } } } ,
9676     empty .value_forbidden:n = true ,
9677     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9678 }
9679
9680
9681 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9682 {
9683   \IfPackageLoadedTF { tikz }
9684   {
9685     \group_begin:
9686     \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.

```

9687     \tl_set:Nn \l_tmpa_tl { { #2 } }
9688     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9689     { \@@_for_a_block:nnnnn ##1 }
9690     \@@_all_the_cells:
9691     \group_end:
9692   }
9693   { \@@_error:n { TikzEveryCell-without-tikz } }
9694 }
9695
9696
9697 \cs_new_protected:Nn \@@_all_the_cells:
9698 {
9699   \int_step_inline:nn \c@iRow
9700   {
9701     \int_step_inline:nn \c@jCol
9702     {
9703       \cs_if_exist:cF { cell - ##1 - #####1 }
9704       {
9705         \clist_if_in:NcF \l_@@_corners_cells_clist
9706         { ##1 - #####1 }
9707         {
9708           \bool_set_false:N \l_tmpa_bool
9709           \cs_if_exist:cTF
9710           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9711           {
9712             \bool_if:NF \l_@@_empty_bool
9713             { \bool_set_true:N \l_tmpa_bool }
9714           }
9715           {
9716             \bool_if:NF \l_@@_not_empty_bool
9717             { \bool_set_true:N \l_tmpa_bool }
9718           }
9719           \bool_if:NT \l_tmpa_bool
9720           {
9721             \@@_block_tikz:nnnnn
9722             \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9723           }
9724         }
9725       }
9726     }
9727   }
9728 }
9729
9730 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9731 {
9732   \bool_if:NF \l_@@_empty_bool
9733   {
9734     \@@_block_tikz:nnnnn

```

```

9735         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9736     }
9737     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9738 }
9739
9740 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9741 {
9742     \int_step_inline:nnn { #1 } { #3 }
9743     {
9744         \int_step_inline:nnn { #2 } { #4 }
9745         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9746     }
9747 }

```

## 37 The command \ShowCellNames

```

9748 \NewDocumentCommand \@@_ShowCellNames { }
9749 {
9750     \bool_if:NT \l_@@_in_code_after_bool
9751     {
9752         \pgfpicture
9753         \pgfrememberpicturepositiononpagetrue
9754         \pgf@relevantforpicturesizefalse
9755         \pgfpathrectanglecorners
9756         { \@@_qpoint:n { 1 } }
9757         {
9758             \@@_qpoint:n
9759             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9760         }
9761         \pgfsetfillopacity { 0.75 }
9762         \pgfsetfillcolor { white }
9763         \pgfusepathqfill
9764         \endpgfpicture
9765     }
9766     \dim_gzero_new:N \g_@@_tmpc_dim
9767     \dim_gzero_new:N \g_@@_tmpd_dim
9768     \dim_gzero_new:N \g_@@_tmpe_dim
9769     \int_step_inline:nn { \c@iRow }
9770     {
9771         \bool_if:NTF \l_@@_in_code_after_bool
9772         {
9773             \pgfpicture
9774             \pgfrememberpicturepositiononpagetrue
9775             \pgf@relevantforpicturesizefalse
9776         }
9777         { \begin { pgfpicture } }
9778         \@@_qpoint:n { row - ##1 }
9779         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9780         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9781         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9782         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9783         \bool_if:NTF \l_@@_in_code_after_bool
9784         { \endpgfpicture }
9785         { \end { pgfpicture } }
9786         \int_step_inline:nn { \c@jCol }
9787         {
9788             \hbox_set:Nn \l_tmpa_box
9789             {
9790                 \normalfont \Large \sffamily \bfseries
9791                 \bool_if:NTF \l_@@_in_code_after_bool
9792                 { \color { red } }
9793                 { \color { red ! 50 } }

```

```

9794         ##1 - ####1
9795     }
9796     \bool_if:NTF \l_@@_in_code_after_bool
9797     {
9798         \pgfpicture
9799         \pgfrememberpicturepositiononpagetrue
9800         \pgf@relevantforpicturesizefalse
9801     }
9802     { \begin { pgfpicture } }
9803     \@@_qpoint:n { col - ####1 }
9804     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9805     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9806     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9807     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9808     \bool_if:NTF \l_@@_in_code_after_bool
9809     { \endpgfpicture }
9810     { \end { pgfpicture } }
9811     \fp_set:Nn \l_tmpa_fp
9812     {
9813         \fp_min:nn
9814         {
9815             \fp_min:nn
9816             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9817             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9818         }
9819         { 1.0 }
9820     }
9821     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9822     \pgfpicture
9823     \pgfrememberpicturepositiononpagetrue
9824     \pgf@relevantforpicturesizefalse
9825     \pgftransformshift
9826     {
9827         \pgfpoint
9828         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9829         { \dim_use:N \g_tmpa_dim }
9830     }
9831     \pgfnode
9832     { rectangle }
9833     { center }
9834     { \box_use:N \l_tmpa_box }
9835     { }
9836     { }
9837     \endpgfpicture
9838 }
9839 }
9840 }

```

## 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.

```
9841 \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.

```
9842 \bool_new:N \g_@@_footnote_bool
```

```

9843 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9844 {
9845   You-have-used-the-key~' \l_keys_key_str '~when-loading-nicematrix~
9846   but-that-key-is-unknown. \\
9847   It-will-be-ignored. \\
9848   For-a-list-of-the-available-keys,~type-H-<return>.
9849 }
9850 {
9851   The-available-keys-are~(in~alphabetic~order):~
9852   footnote,~
9853   footnotehyper,~
9854   messages-for-Overleaf,~
9855   renew-dots-and~
9856   renew-matrix.
9857 }
9858 \keys_define:nn { nicematrix }
9859 {
9860   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9861   renew-dots .value_forbidden:n = true ,
9862   renew-matrix .code:n = \@@_renew_matrix: ,
9863   renew-matrix .value_forbidden:n = true ,
9864   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9865   footnote .bool_set:N = \g_@@_footnote_bool ,
9866   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9867   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9868 }
9869 \ProcessKeyOptions

9870 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9871 {
9872   You-can't-use-the-option~'footnote'~because-the-package~
9873   footnotehyper~has-already-been-loaded.~
9874   If-you-want,~you-can-use-the-option~'footnotehyper'~and-the-footnotes~
9875   within~the~environments~of~nicematrix~will-be-extracted-with-the-tools~
9876   of~the~package~footnotehyper.\\
9877   The-package~footnote~won't-be-loaded.
9878 }
9879 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9880 {
9881   You-can't-use-the-option~'footnotehyper'~because-the-package~
9882   footnote~has-already-been-loaded.~
9883   If-you-want,~you-can-use-the-option~'footnote'~and-the-footnotes~
9884   within~the~environments~of~nicematrix~will-be-extracted-with-the-tools~
9885   of~the~package~footnote.\\
9886   The-package~footnotehyper~won't-be-loaded.
9887 }

9888 \bool_if:NT \g_@@_footnote_bool
9889 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9890 \IfClassLoadedTF { beamer }
9891 { \bool_set_false:N \g_@@_footnote_bool }
9892 {
9893   \IfPackageLoadedTF { footnotehyper }
9894   { \@@_error:n { footnote-with-footnotehyper-package } }
9895   { \usepackage { footnote } }
9896 }
9897 }
9898 \bool_if:NT \g_@@_footnotehyper_bool
9899 {

```



The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9900 \IfClassLoadedTF { beamer }
9901   { \bool_set_false:N \g_@@_footnote_bool }
9902   {
9903     \IfPackageLoadedTF { footnote }
9904       { \@@_error:n { footnotehyper~with~footnote~package } }
9905       { \usepackage { footnotehyper } }
9906   }
9907   \bool_set_true:N \g_@@_footnote_bool
9908 }

```

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.

```

9909 \bool_new:N \l_@@_underscore_loaded_bool
9910 \IfPackageLoadedT { underscore }
9911   { \bool_set_true:N \l_@@_underscore_loaded_bool }
9912 \hook_gput_code:nnn { begindocument } { . }
9913 {
9914   \bool_if:NF \l_@@_underscore_loaded_bool
9915   {
9916     \IfPackageLoadedT { underscore }
9917       { \@@_error:n { underscore~after~nicematrix } }
9918   }
9919 }

```

## 40 Error messages of the package

When there is a unknown key, we try a “normal form” of the key and, when that normal form exists, we add that information in the error message.

The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of `nicematrix`).

`#1` is a clist of names of sets of keys and `#2` is the error message to send.

```

9920 \cs_new_protected:Npn \@@_unknown_key:nn #1 #2
9921   {
9922     \str_set_eq:NN \l_tmpa_str \l_keys_key_str
9923     \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
9924     \str_set:Nx \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
9925     \bool_set_false:N \l_tmpa_bool
9926     \clist_map_inline:nn { #1 }
9927       {
9928         \keys_if_exist:neT { ##1 } { \l_tmpa_str }
9929         {
9930           \@@_error:n { key~with~normal~form~exists }
9931           \bool_set_true:N \l_tmpa_bool
9932           \clist_map_break:
9933         }
9934       }
9935   }

```

```

9934     }
9935     \bool_if:NF \l_tmpa_bool { \@@_error:n { #2 } }
9936   }
9937   \@@_msg_new:nn { key-with-normal-form-exists }
9938   {
9939     The-key~'\l_keys_key_str'~does-not-exists.\\
9940     It~will-be-ignored.\\
9941     Maybe~you-want~to-use~the~key~'\l_tmpa_str'.
9942   }
9943   \str_const:Ne \c_@@_available_keys_str
9944   {
9945     \bool_if:Nt { ! \g_@@_messages_for_Overleaf_bool }
9946     { For~a~list~of~the~available~keys,~type-H~<return>. }
9947   }
9948   \seq_new:N \g_@@_types_of_matrix_seq
9949   \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9950   {
9951     NiceMatrix ,
9952     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9953   }
9954   \seq_gset_map:e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9955   { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_err_too_many_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9956 \cs_new_protected:Npn \@@_err_too_many_cols:
9957 {
9958   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9959   { \@@_fatal:nn { too-many-cols-for-array } }
9960   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9961   { \@@_fatal:n { too-many-cols-for-matrix } }
9962   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9963   { \@@_fatal:n { too-many-cols-for-matrix } }
9964   \bool_if:NF \l_@@_last_col_without_value_bool
9965   { \@@_fatal:n { too-many-cols-for-matrix-with-last-col } }
9966 }

```

The following command must *not* be protected since it's used in an error message.

```

9967 \cs_new:Npn \@@_message_hdotsfor:
9968 {
9969   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9970   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9971     \token_to_str:N \Hbrace \ is~incorrect. }
9972 }
9973 \cs_new_protected:Npn \@@_Hline_in_cell:
9974 { \@@_fatal:n { Misuse-of-Hline } }
9975 \@@_msg_new:nn { Misuse-of-Hline }
9976 {
9977   Misuse~of~Hline. \\
9978   Error~in~your~row~ \int_eval:n { \c@iRow }. \\
9979   \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
9980   That~error~is~fatal.
9981 }
9982 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9983 {
9984   Incompatible~options.\\
9985   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
9986   The~output~will~not~be~reliable.
9987 }

```

```

9988 \@@_msg_new:nn { Body-alone }
9989 {
9990   \token_to_str:N \Body\ alone. \\
9991   You-have-used-\token_to_str:N \Body\ without-\token_to_str:N \CodeBefore.\\
9992   That~error~is~fatal.
9993 }
9994 \@@_msg_new:nn { cellcolor-in-Block }
9995 {
9996   Bad-use-of-\token_to_str:N \cellcolor \\
9997   You-can't-use-\token_to_str:N \cellcolor\ in-\token_to_str:N \Block\
9998   (except~in~a~sub~block).\\
9999   That~command~will~be~ignored.
10000 }
10001 \@@_msg_new:nn { rowcolor-in-Block }
10002 {
10003   Bad-use-of-\token_to_str:N \rowcolor \\
10004   You-can't-use-\token_to_str:N \rowcolor\ in-\token_to_str:N \Block.\\
10005   That~command~will~be~ignored.
10006 }
10007 \@@_msg_new:nn { key-color-inside }
10008 {
10009   Deleted~key.\\
10010   The~key~'color-inside'~(and~its~alias~'colortbl-like')~has~been~deleted~in
10011   ~'nicematrix'~and~must~not~be~used.\\
10012   This~error~is~fatal.
10013 }
10014 \@@_msg_new:nn { invalid-weight }
10015 {
10016   Unknown~key.\\
10017   The~key~' \l_keys_key_str '~-of~your~column~X~is~unknown~and~will~be~ignored.
10018 }
10019 \@@_msg_new:nn { last-col-not-used }
10020 {
10021   Column~not~used.\\
10022   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
10023   in~your~\@@_full_name_env: .~
10024   However,~you~can~go~on.
10025 }
10026 \@@_msg_new:nn { too-many-cols-for-matrix-with-last-col }
10027 {
10028   Too-many-columns.\\
10029   In~the~row~ \int_eval:n { \c@iRow },~
10030   you~try~to~use~more~columns~
10031   than~allowed~by~your~ \@@_full_name_env: .
10032   \@@_message_hdotsfor: \
10033   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
10034   (plus~the~exterior~columns).~This~error~is~fatal.
10035 }
10036 \@@_msg_new:nn { too-many-cols-for-matrix }
10037 {
10038   Too-many-columns.\\
10039   In~the~row~ \int_eval:n { \c@iRow },~
10040   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
10041   \@@_message_hdotsfor: \
10042   Recall~that~the~maximal~number~of~columns~for~a~matrix~
10043   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
10044   LaTeX~counter~'MaxMatrixCols'.~
10045   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
10046   (use~ \token_to_str:N \setcounter \ to~change~that~value).~
10047   This~error~is~fatal.
10048 }

```

```

10049 \@@_msg_new:nn { too-many-cols-for-array }
10050 {
10051   Too-many-columns.\
10052   In-the-row~ \int_eval:n { \c@iRow } ,~
10053   ~you~try~to~use~more~columns~than~allowed~by~your~
10054   \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
10055   \int_use:N \g_@@_static_num_of_col_int \
10056   \bool_if:nT
10057     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
10058     { (plus~the~exterior~ones)~}
10059   since~the~preamble~is~' \g_@@_user_preamble_tl '.\
10060   This~error~is~fatal.
10061 }
10062 \@@_msg_new:nn { columns~not~used }
10063 {
10064   Columns~not~used.\
10065   The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
10066   It~announces~ \int_use:N \g_@@_static_num_of_col_int \
10067   columns~but~you~only~used~ \int_use:N \c@jCol .\
10068   The~columns~you~did~not~used~won't~be~created.\
10069   You~won't~have~similar~warning~till~the~end~of~the~document.
10070 }
10071 \@@_msg_new:nn { empty~preamble }
10072 {
10073   Empty~preamble.\
10074   The~preamble~of~your~ \@@_full_name_env: \ is~empty.\
10075   This~error~is~fatal.
10076 }
10077 \@@_msg_new:nn { in~first~col }
10078 {
10079   Erroneous~use.\
10080   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
10081   That~command~will~be~ignored.
10082 }
10083 \@@_msg_new:nn { in~last~col }
10084 {
10085   Erroneous~use.\
10086   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
10087   That~command~will~be~ignored.
10088 }
10089 \@@_msg_new:nn { in~first~row }
10090 {
10091   Erroneous~use.\
10092   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\
10093   That~command~will~be~ignored.
10094 }
10095 \@@_msg_new:nn { in~last~row }
10096 {
10097   Erroneous~use.\
10098   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\
10099   That~command~will~be~ignored.
10100 }
10101 \@@_msg_new:nn { TopRule~without~booktabs }
10102 {
10103   Erroneous~use.\
10104   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\
10105   That~command~will~be~ignored.
10106 }
10107 \@@_msg_new:nn { TopRule~without~tikz }
10108 {

```

```

10109     Erroneous~use.\\
10110     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
10111     That~command~will~be~ignored.
10112 }
10113 \@@_msg_new:nn { caption~outside~float }
10114 {
10115     Key~caption~forbidden.\\
10116     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
10117     environment~(such~as~\{table\}).~This~key~will~be~ignored.
10118 }
10119 \@@_msg_new:nn { short~caption~without~caption }
10120 {
10121     You~should~not~use~the~key~'short~caption'~without~'caption'.~
10122     However,~your~'short~caption'~will~be~used~as~'caption'.
10123 }
10124 \@@_msg_new:nn { double~closing~delimiter }
10125 {
10126     Double~delimiter.\\
10127     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
10128     delimiter.~This~delimiter~will~be~ignored.
10129 }
10130 \@@_msg_new:nn { delimiter~after~opening }
10131 {
10132     Double~delimiter.\\
10133     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
10134     delimiter.~That~delimiter~will~be~ignored.
10135 }
10136 \@@_msg_new:nn { bad~option~for~line~style }
10137 {
10138     Bad~line~style.\\
10139     Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line~style'~
10140     is~'standard'.~That~key~will~be~ignored.
10141 }
10142 \@@_msg_new:nn { corners~with~no~cell~nodes }
10143 {
10144     Incompatible~keys.\\
10145     You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
10146     is~in~force.\\
10147     If~you~go~on,~that~key~will~be~ignored.
10148 }
10149 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
10150 {
10151     Incompatible~keys.\\
10152     You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
10153     is~in~force.\\
10154     If~you~go~on,~those~extra~nodes~won't~be~created.
10155 }
10156 \@@_msg_new:nn { Identical~notes~in~caption }
10157 {
10158     Identical~tabular~notes.\\
10159     You~can't~put~several~notes~with~the~same~content~in~
10160     \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
10161     If~you~go~on,~the~output~will~probably~be~erroneous.
10162 }
10163 \@@_msg_new:nn { tabularnote~below~the~tabular }
10164 {
10165     \token_to_str:N \tabularnote \ forbidden\\
10166     You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
10167     of~your~tabular~because~the~caption~will~be~composed~below~
10168     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~

```

```

10169     key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .\\
10170     Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
10171     no~similar~error~will~raised~in~this~document.
10172   }

10173   \@@_msg_new:nn { Unknown~key~for~rules }
10174   {
10175     Unknown~key.\\
10176     There~is~only~two~keys~available~here:~width~and~color.\\
10177     Your~key~' \l_keys_key_str '~will~be~ignored.
10178   }

10179   \@@_msg_new:nn { Unknown~key~for~Hbrace }
10180   {
10181     Unknown~key.\\
10182     You~have~used~the~key~' \l_keys_key_str '~but~the~only~
10183     keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
10184     and~ \token_to_str:N \Vbrace \ are:~'brace-shift',~'color',~
10185     'horizontal-label(s)',~'shorten'~'shorten-end'~
10186     and~'shorten-start'.\\
10187     That~error~is~fatal.
10188   }

10189   \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
10190   {
10191     Unknown~key.\\
10192     There~is~only~two~keys~available~here:~
10193     'empty'~and~'not-empty'.\\
10194     Your~key~' \l_keys_key_str '~will~be~ignored.
10195   }

10196   \@@_msg_new:nn { Unknown~key~for~rotate }
10197   {
10198     Unknown~key.\\
10199     The~only~key~available~here~is~'c'.\\
10200     Your~key~' \l_keys_key_str '~will~be~ignored.
10201   }

10202   \@@_msg_new:nnn { Unknown~key~for~custom-line }
10203   {
10204     Unknown~key.\\
10205     The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
10206     It~you~go~on,~you~will~probably~have~other~errors. \\
10207     \c_@@_available_keys_str
10208   }
10209   {
10210     The~available~keys~are~(in~alphabetic~order):~
10211     ccommand,~
10212     color,~
10213     command,~
10214     dotted,~
10215     letter,~
10216     multiplicity,~
10217     sep-color,~
10218     tikz,~and~total-width.
10219   }

10220   \@@_msg_new:nnn { Unknown~key~for~xdots }
10221   {
10222     Unknown~key.\\
10223     The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
10224     \c_@@_available_keys_str
10225   }
10226   {
10227     The~available~keys~are~(in~alphabetic~order):~
10228     'color',~
10229     'horizontal(s)-labels',~

```

```

10230     'inter',~
10231     'line-style',~
10232     'nullify',~
10233     'radius',~
10234     'shorten',~
10235     'shorten-end'~and~'shorten-start'.
10236 }

10237 \@@_msg_new:nn { Unknown~key~for~rowcolors }
10238 {
10239     Unknown~key.\\
10240     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
10241     (and~you~try~to~use~' \l_keys_key_str ')\
10242     That~key~will~be~ignored.
10243 }

10244 \@@_msg_new:nn { label~without~caption }
10245 {
10246     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
10247     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
10248 }

10249 \@@_msg_new:nn { W~warning }
10250 {
10251     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
10252     (row~ \int_use:N \c@iRow ).
10253 }

10254 \@@_msg_new:nn { Construct~too~large }
10255 {
10256     Construct~too~large.\\
10257     Your~command~ \token_to_str:N #1
10258     can't~be~drawn~because~your~matrix~is~too~small.\\
10259     That~command~will~be~ignored.
10260 }

10261 \@@_msg_new:nn { underscore~after~nicematrix }
10262 {
10263     Problem~with~'underscore'.\\
10264     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
10265     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
10266     ' \token_to_str:N \Cdots \token_to_str:N _
10267     \{ n \token_to_str:N \text \{ ~times \} \}'.
10268 }

10269 \@@_msg_new:nn { ampersand~in~light-syntax }
10270 {
10271     Ampersand~forbidden.\\
10272     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
10273     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
10274 }

10275 \@@_msg_new:nn { double~backslash~in~light-syntax }
10276 {
10277     Double~backslash~forbidden.\\
10278     You~can't~use~ \token_to_str:N \
10279     ~to~separate~rows~because~the~key~'light-syntax'~
10280     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
10281     (set~by~the~key~'end-of-row').~This~error~is~fatal.
10282 }

10283 \@@_msg_new:nn { hlines~with~color }
10284 {
10285     Incompatible~keys.\\
10286     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
10287     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
10288     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
10289     Your~key~will~be~discarded.
10290 }

```

```

10291 \@@_msg_new:nn { bad-value-for-baseline }
10292 {
10293   Bad-value-for-baseline.\
10294   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10295   valid.~The-value-must-be-between-\int_use:N \l_@@_first_row_int\ and~
10296   \int_use:N \g_@@_row_total_int \ or-equal-to-'t',~'c'~or~'b'~or-of~
10297   the-form-'line-i'.\
10298   A-value-of-1~will-be-used.
10299 }
10300 \@@_msg_new:nn { bad-value-for-baseline-line }
10301 {
10302   Bad-value-for-baseline-with-line.\
10303   The-value-given-to-'baseline'~( \int_use:N \l_tmpa_int )~is-not~
10304   valid.~The-number-of-the-line-must-be-between-1~and~
10305   \int_eval:n { \c@iRow + 1 } \
10306   A-value-of-'line-1'~will-be-used.
10307 }
10308 \@@_msg_new:nn { detection-of-empty-cells }
10309 {
10310   Problem-with-'not-empty'\\
10311   For-technical-reasons,~you-must-activate~
10312   'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
10313   in-order-to-use-the-key~' \l_keys_key_str '.\
10314   That-key-will-be-ignored.
10315 }
10316 \@@_msg_new:nn { siunitx-not-loaded }
10317 {
10318   siunitx-not-loaded\\
10319   You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\
10320   That-error-is-fatal.
10321 }
10322 \@@_msg_new:nn { Invalid-name }
10323 {
10324   Invalid-name.\
10325   You-can't-give-the-name-' \l_keys_value_tl '~to-a~ \token_to_str:N
10326   \SubMatrix \ of-your~ \@@_full_name_env: .\
10327   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
10328   This-key-will-be-ignored.
10329 }
10330 \@@_msg_new:nn { Hbrace-not-allowed }
10331 {
10332   Command-not-allowed.\
10333   You-can't-use-the-command~ \token_to_str:N #1
10334   because-you-have-not-loaded~
10335   \IfPackageLoadedTF { tikz }
10336   { the-TikZ-library~'decorations.pathreplacing'~Use~ }
10337   { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10338   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \
10339   That-command-will-be-ignored.
10340 }
10341 \@@_msg_new:nn { Vbrace-not-allowed }
10342 {
10343   Command-not-allowed.\
10344   You-can't-use-the-command~ \token_to_str:N \Vbrace \
10345   because-you-have-not-loaded-TikZ~
10346   and-the-TikZ-library~'decorations.pathreplacing'.\
10347   Use: ~\token_to_str:N \usepackage \{tikz\}~
10348   \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \
10349   That-command-will-be-ignored.
10350 }
10351 \@@_msg_new:nn { Wrong-line-in-SubMatrix }

```



```

10352 {
10353     Wrong~line.\\
10354     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10355     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10356     number~is~not~valid.~It~will~be~ignored.
10357 }
10358 \@@_msg_new:nn { Impossible~delimiter }
10359 {
10360     Impossible~delimiter.\\
10361     It's~impossible~to~draw~the~#1~delimiter~of~your~
10362     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10363     in~that~column.
10364     \bool_if:NT \l_@@_submatrix_slim_bool
10365     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10366     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10367 }
10368 \@@_msg_new:nnn { width~without~X~columns }
10369 {
10370     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10371     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10372     That~key~will~be~ignored.
10373 }
10374 {
10375     This~message~is~the~message~'width~without~X~columns'~
10376     of~the~module~'nicematrix'.~
10377     The~experimented~users~can~disable~that~message~with~
10378     \token_to_str:N \msg_redirect_name:nnn .\\
10379 }
10380
10381 \@@_msg_new:nn { key~multiplicity~with~dotted }
10382 {
10383     Incompatible~keys. \\
10384     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10385     in~a~'custom~line'.~They~are~incompatible. \\
10386     The~key~'multiplicity'~will~be~discarded.
10387 }
10388 \@@_msg_new:nn { empty~environment }
10389 {
10390     Empty~environment.\\
10391     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10392 }
10393 \@@_msg_new:nn { No~letter~and~no~command }
10394 {
10395     Erroneous~use.\\
10396     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10397     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10398     '~'ccommand'~(to~draw~horizontal~rules).\\
10399     However,~you~can~go~on.
10400 }
10401 \@@_msg_new:nn { Forbidden~letter }
10402 {
10403     Forbidden~letter.\\
10404     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10405     It~will~be~ignored.\\
10406     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10407 }
10408 \@@_msg_new:nn { Several~letters }
10409 {
10410     Wrong~name.\\
10411     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10412     have~used~' \l_@@_letter_str ').\\

```

```

10413     It~will~be~ignored.
10414 }
10415 \@@_msg_new:nn { Delimiter-with-small }
10416 {
10417     Delimiter~forbidden.\\
10418     You~can't~put~a~delimiter~in~the~preamble~of~your~
10419     \@@_full_name_env: \
10420     because~the~key~'small'~is~in~force.\\
10421     This~error~is~fatal.
10422 }
10423 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10424 {
10425     Unknown~cell.\\
10426     Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10427     the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10428     can't~be~executed~because~a~cell~doesn't~exist.\\
10429     This~command~ \token_to_str:N \line \ will~be~ignored.
10430 }
10431 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10432 {
10433     Duplicate~name.\\
10434     The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10435     in~this~ \@@_full_name_env: .\\
10436     This~key~will~be~ignored.\\
10437     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10438     { For~a~list~of~the~names~already~used,~type~H<return>. }
10439 }
10440 {
10441     The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10442     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10443 }
10444 \@@_msg_new:nn { r~or~l~with~preamble }
10445 {
10446     Erroneous~use.\\
10447     You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10448     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10449     your~ \@@_full_name_env: .\\
10450     This~key~will~be~ignored.
10451 }
10452 \@@_msg_new:nn { Hdotsfor~in~col=0 }
10453 {
10454     Erroneous~use.\\
10455     You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
10456     in~an~exterior~column~of~
10457     the~array.~This~error~is~fatal.
10458 }
10459 \@@_msg_new:nn { bad~corner }
10460 {
10461     Bad~corner.\\
10462     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10463     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10464     This~specification~of~corner~will~be~ignored.
10465 }
10466 \@@_msg_new:nn { bad~border }
10467 {
10468     Bad~border.\\
10469     \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10470     (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10471     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10472     also~use~the~key~'tikz'
10473     \IfPackageLoadedF { tikz }

```

```

10474     { ~if-you-load-the-LaTeX-package~'tikz' } ).\\
10475     This~specification-of~border~will~be~ignored.
10476 }

10477 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10478 {
10479     TikZ~not~loaded.\\
10480     You~can't~use~ \token_to_str:N \TikzEveryCell \
10481     because~you~have~not~loaded~tikz.~
10482     This~command~will~be~ignored.
10483 }

10484 \@@_msg_new:nn { tikz~key~without~tikz }
10485 {
10486     TikZ~not~loaded.\\
10487     You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
10488     \Block '~because~you~have~not~loaded~tikz.~
10489     This~key~will~be~ignored.
10490 }

10491 \@@_msg_new:nn { Bad~argument~for~Block }
10492 {
10493     Bad~argument.\\
10494     The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
10495     be~of~the~form~'i-j'~(or~completely~empty)~and~you~have~used:~
10496     '#1'. \\
10497     If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono~cell~(as~if~
10498     the~argument~was~empty).
10499 }

10500 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10501 {
10502     Erroneous~use.\\
10503     In~the~ \@@_full_name_env: ,~you~must~use~the~key~
10504     'last~col'~without~value.\\
10505     However,~you~can~go~on~for~this~time~
10506     (the~value~' \l_keys_value_tl '~will~be~ignored).
10507 }

10508 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
10509 {
10510     Erroneous~use. \\
10511     In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
10512     'last~col'~without~value. \\
10513     However,~you~can~go~on~for~this~time~
10514     (the~value~' \l_keys_value_tl '~will~be~ignored).
10515 }

10516 \@@_msg_new:nn { Block~too~large~1 }
10517 {
10518     Block~too~large. \\
10519     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10520     too~small~for~that~block. \\
10521     This~block~and~maybe~others~will~be~ignored.
10522 }

10523 \@@_msg_new:nn { Block~too~large~2 }
10524 {
10525     Block~too~large. \\
10526     The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
10527     \g_@@_static_num_of_col_int \
10528     columns~but~you~use~only~ \int_use:N \c_jCol \ and~that's~why~a~block~
10529     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10530     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10531     This~block~and~maybe~others~will~be~ignored.
10532 }

10533 \@@_msg_new:nn { unknown~column~type }

```

```

10534 {
10535     Bad-column-type. \\
10536     The-column-type~'#1'~in~your~ \@@_full_name_env: \
10537     is~unknown. \\
10538     This~error~is~fatal.
10539 }
10540 \@@_msg_new:nn { unknown~column-type-multicolumn }
10541 {
10542     Bad-column-type. \\
10543     The-column-type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10544     ~of~your~ \@@_full_name_env: \
10545     is~unknown. \\
10546     This~error~is~fatal.
10547 }
10548 \@@_msg_new:nn { unknown~column-type-S }
10549 {
10550     Bad-column-type. \\
10551     The-column-type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10552     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10553     load~that~package. \\
10554     This~error~is~fatal.
10555 }
10556 \@@_msg_new:nn { unknown~column-type-S~multicolumn }
10557 {
10558     Bad-column-type. \\
10559     The-column-type~'S'~in~the~command~\token_to_str:N \multicolumn \
10560     of~your~ \@@_full_name_env: \ is~unknown. \\
10561     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10562     load~that~package. \\
10563     This~error~is~fatal.
10564 }
10565 \@@_msg_new:nn { tabularnote~forbidden }
10566 {
10567     Forbidden~command. \\
10568     You~can't~use~the~command~ \token_to_str:N \tabularnote \
10569     ~here.~This~command~is~available~only~in~
10570     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10571     the~argument~of~a~command~\token_to_str:N \caption \ included~
10572     in~an~environment~\{table\}. \\
10573     This~command~will~be~ignored.
10574 }
10575 \@@_msg_new:nn { borders~forbidden }
10576 {
10577     Forbidden~key.\\
10578     You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10579     because~the~option~'rounded-corners'~
10580     is~in~force~with~a~non-zero~value.\\
10581     This~key~will~be~ignored.
10582 }
10583 \@@_msg_new:nn { bottomrule~without~booktabs }
10584 {
10585     booktabs~not~loaded.\\
10586     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10587     loaded~'booktabs'.\\
10588     This~key~will~be~ignored.
10589 }
10590 \@@_msg_new:nn { enumitem~not~loaded }
10591 {
10592     enumitem~not~loaded. \\
10593     You~can't~use~the~command~ \token_to_str:N \tabularnote \
10594     ~because~you~haven't~loaded~'enumitem'. \\

```

```

10595 All-the-commands~ \token_to_str:N \tabularnote \ will-be-
10596 ignored-in-the-document.
10597 }
10598 \@@_msg_new:nn { tikz-without-tikz }
10599 {
10600 TikZ-not-loaded. \\
10601 You-can't-use-the-key~'tikz'~here~because~TikZ-is-not~
10602 loaded.~If-you-go-on,~that-key~will-be-ignored.
10603 }
10604 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
10605 {
10606 TikZ-not-loaded. \\
10607 You-have-used-the-key~'tikz'~in-the-definition-of~a~
10608 customized~line~(with~'custom-line')~but~TikZ-is-not-loaded.~
10609 You-can-go-on-but-you-will-have-another~error~if-you-actually~
10610 use~that~custom~line.
10611 }
10612 \@@_msg_new:nn { tikz-in-borders-without-tikz }
10613 {
10614 TikZ-not-loaded. \\
10615 You-have-used-the-key~'tikz'~in~a~key~'borders'~(of~a~
10616 command~' \token_to_str:N \Block ')~but~TikZ-is-not-loaded.~
10617 That~key~will-be-ignored.
10618 }
10619 \@@_msg_new:nn { color-in-custom-line-with-tikz }
10620 {
10621 Erroneous~use.\\
10622 In~a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
10623 which-is-forbidden~(you-should-use~'color'~inside~the-key~'tikz').~
10624 The-key~'color'~will-be-discarded.
10625 }
10626 \@@_msg_new:nn { Wrong-last-row }
10627 {
10628 Wrong~number.\\
10629 You-have-used~'last-row= \int_use:N \l_@@_last_row_int '~but-your~
10630 \@@_full_name_env: \ seems-to-have~ \int_use:N \c@iRow \ rows.~
10631 If-you-go-on,~the-value-of~ \int_use:N \c@iRow \ will-be-used-for~
10632 last-row-but-you-should-correct-your-code.~You-can-avoid-this~
10633 problem-by-using~'last-row'~without~value~(more-compilations~
10634 might-be-necessary).
10635 }
10636 \@@_msg_new:nn { Yet-in-env }
10637 {
10638 Nested~environments.\\
10639 Environments-of~nicematrix~can't-be-nested.\\
10640 This-error-is-fatal.
10641 }
10642 \@@_msg_new:nn { Outside-math-mode }
10643 {
10644 Outside~math~mode.\\
10645 The~\@@_full_name_env: \ can-be-used-only-in-math-mode~
10646 (and-not-in~ \token_to_str:N \vcenter ).\\
10647 This-error-is-fatal.
10648 }
10649 \@@_msg_new:nn { One-letter-allowed }
10650 {
10651 Bad~name.\\
10652 The-value-of-key~' \l_keys_key_str '~must-be-of-length~1~and~
10653 you-have-used~' \l_keys_value_tl '~.\\
10654 It~will-be-ignored.
10655 }

```

```

10656 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10657 {
10658   Environment~\{TabularNote\}~forbidden.\\
10659   You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10660   but~*before*~the~ \token_to_str:N \CodeAfter . \\
10661   This~environment~\{TabularNote\}~will~be~ignored.
10662 }
10663 \@@_msg_new:nn { varwidth~not~loaded }
10664 {
10665   varwidth~not~loaded.\\
10666   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10667   loaded.\\
10668   Your~column~will~behave~like~'p'.
10669 }
10670 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10671 {
10672   varwidth~not~loaded.\\
10673   You~can't~use~the~key~'V'~in~your~column~'X'~
10674   because~'varwidth'~is~not~loaded.\\
10675   It~will~be~ignored. \\
10676 }
10677 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10678 {
10679   Unknown~key.\\
10680   Your~key~' \l_keys_key_str ' ~is~unknown~for~a~rule.\\
10681   \c_@@_available_keys_str
10682 }
10683 {
10684   The~available~keys~are~(in~alphabetic~order):~
10685   color,~
10686   dotted,~
10687   multiplicity,~
10688   sep~color,~
10689   tikz,~and~total~width.
10690 }
10691
10692 \@@_msg_new:nnn { Unknown~key~for~Block }
10693 {
10694   Unknown~key. \\
10695   The~key~' \l_keys_key_str ' ~is~unknown~for~the~command~
10696   \token_to_str:N \Block . \\
10697   It~will~be~ignored. \\
10698   \c_@@_available_keys_str
10699 }
10700 {
10701   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10702   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10703   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10704   and~vlines.
10705 }
10706 \@@_msg_new:nnn { Unknown~key~for~Brace }
10707 {
10708   Unknown~key.\\
10709   The~key~' \l_keys_key_str ' ~is~unknown~for~the~commands~
10710   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10711   It~will~be~ignored. \\
10712   \c_@@_available_keys_str
10713 }
10714 {
10715   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10716   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10717   right~shorten)~and~yshift.

```

```

10718 }
10719 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10720 {
10721   Unknown~key.\\
10722   The~key~' \l_keys_key_str '~is~unknown.\\
10723   It~will~be~ignored. \\
10724   \c_@@_available_keys_str
10725 }
10726 {
10727   The~available~keys~are~(in~alphabetic~order):~
10728   delimiters/color,~
10729   rules~(with~the~subkeys~'color'~and~'width'),~
10730   sub-matrix~(several~subkeys)~
10731   and~xdots~(several~subkeys).~
10732   The~latter~is~for~the~command~ \token_to_str:N \line .
10733 }
10734 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10735 {
10736   Unknown~key.\\
10737   The~key~' \l_keys_key_str '~is~unknown.\\
10738   It~will~be~ignored. \\
10739   \c_@@_available_keys_str
10740 }
10741 {
10742   The~available~keys~are~(in~alphabetic~order):~
10743   create-cell-nodes,~
10744   delimiters/color~and~
10745   sub-matrix~(several~subkeys).
10746 }
10747 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10748 {
10749   Unknown~key.\\
10750   The~key~' \l_keys_key_str '~is~unknown.\\
10751   That~key~will~be~ignored. \\
10752   \c_@@_available_keys_str
10753 }
10754 {
10755   The~available~keys~are~(in~alphabetic~order):~
10756   'delimiters/color',~
10757   'extra-height',~
10758   'hlines',~
10759   'hvlines',~
10760   'left-xshift',~
10761   'name',~
10762   'right-xshift',~
10763   'rules'~(with~the~subkeys~'color'~and~'width'),~
10764   'slim',~
10765   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10766   and~'right-xshift').\\
10767 }
10768 \@@_msg_new:nnn { Unknown~key~for~notes }
10769 {
10770   Unknown~key.\\
10771   The~key~' \l_keys_key_str '~is~unknown.\\
10772   That~key~will~be~ignored. \\
10773   \c_@@_available_keys_str
10774 }
10775 {
10776   The~available~keys~are~(in~alphabetic~order):~
10777   bottomrule,~
10778   code-after,~
10779   code-before(+),~

```

```

10780     detect-duplicates,~
10781     enumitem-keys,~
10782     enumitem-keys-para,~
10783     para,~
10784     label-in-list,~
10785     label-in-tabular~and~
10786     style.
10787 }

10788 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10789 {
10790     Unknown~key.\\
10791     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10792     \token_to_str:N \RowStyle . \\
10793     That~key~will~be~ignored. \\
10794     \c_@@_available_keys_str
10795 }
10796 {
10797     The~available~keys~are~(in~alphabetic~order):~
10798     bold,~
10799     cell-space-top-limit,~
10800     cell-space-bottom-limit,~
10801     cell-space-limits,~
10802     color,~
10803     fill~(alias:~rowcolor),~
10804     nb-rows,~
10805     opacity~and~
10806     rounded-corners.
10807 }

10808 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10809 {
10810     Unknown~key.\\
10811     The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10812     \token_to_str:N \NiceMatrixOptions . \\
10813     That~key~will~be~ignored. \\
10814     \c_@@_available_keys_str
10815 }
10816 {
10817     The~available~keys~are~(in~alphabetic~order):~
10818     &-in-blocks,~
10819     allow-duplicate-names,~
10820     ampersand-in-blocks,~
10821     caption-above,~
10822     cell-space-bottom-limit(+),~
10823     cell-space-limits(+),~
10824     cell-space-top-limit(+),~
10825     code-for-first-col,~
10826     code-for-first-row,~
10827     code-for-last-col,~
10828     code-for-last-row,~
10829     corners,~
10830     custom-key,~
10831     create-extra-nodes,~
10832     create-medium-nodes,~
10833     create-large-nodes,~
10834     custom-line,~
10835     delimiters~(several~subkeys),~
10836     end-of-row,~
10837     first-col,~
10838     first-row,~
10839     hlines,~
10840     hvlines,~
10841     hvlines-except-borders,~
10842     last-col,~

```



```

10843     last-row,~
10844     left-margin,~
10845     light-syntax,~
10846     light-syntax-expanded,~
10847     matrix/columns-type,~
10848     no-cell-nodes,~
10849     notes~(several~subkeys),~
10850     nullify-dots,~
10851     pgf-node-code,~
10852     renew-dots,~
10853     renew-matrix,~
10854     respect-arraystretch,~
10855     rounded-corners,~
10856     right-margin,~
10857     rules~(with~the~subkeys~'color'~and~'width'),~
10858     small,~
10859     sub-matrix~(several~subkeys),~
10860     vlines,~
10861     xdots~(several~subkeys).
10862 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10863 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10864 {
10865     Unknown~key.\\
10866     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10867     \{NiceArray\}. \\
10868     That~key~will~be~ignored. \\
10869     \c_@@_available_keys_str
10870 }
10871 {
10872     The~available~keys~are~(in~alphabetic~order):~
10873     &~in~blocks,~
10874     ampersand~in~blocks,~
10875     b,~
10876     baseline,~
10877     c,~
10878     cell-space-bottom-limit,~
10879     cell-space-limits,~
10880     cell-space-top-limit,~
10881     code~after,~
10882     code~for~first~col,~
10883     code~for~first~row,~
10884     code~for~last~col,~
10885     code~for~last~row,~
10886     columns~width,~
10887     corners,~
10888     create~extra~nodes,~
10889     create~medium~nodes,~
10890     create~large~nodes,~
10891     extra~left~margin,~
10892     extra~right~margin,~
10893     first~col,~
10894     first~row,~
10895     hlines,~
10896     hvlines,~
10897     hvlines~except~borders,~
10898     last~col,~
10899     last~row,~
10900     left~margin,~
10901     light~syntax,~
10902     light~syntax~expanded,~
10903     name,~

```

```

10904 no-cell-nodes,~
10905 nullify-dots,~
10906 pgf-node-code,~
10907 renew-dots,~
10908 respect-arraystretch,~
10909 right-margin,~
10910 rounded-corners,~
10911 rules~(with~the~subkeys~'color'~and~'width'),~
10912 small,~
10913 t,~
10914 vlines,~
10915 xdots/color,~
10916 xdots/shorten-start(+),~
10917 xdots/shorten-end(+),~
10918 xdots/shorten(+),~and~
10919 xdots/line-style.
10920 }

```

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`).

```

10921 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10922 {
10923   Unknown~key.\\
10924   The~key~' \l_keys_key_str 'is~unknown~for~the~
10925   \@@_full_name_env: . \\
10926   That~key~will~be~ignored. \\
10927   \c_@@_available_keys_str
10928 }
10929 {
10930   The~available~keys~are~(in~alphabetic~order):~
10931   &-in-blocks,~
10932   ampersand-in-blocks,~
10933   b,~
10934   baseline,~
10935   c,~
10936   cell-space-bottom-limit,~
10937   cell-space-limits,~
10938   cell-space-top-limit,~
10939   code-after,~
10940   code-for-first-col,~
10941   code-for-first-row,~
10942   code-for-last-col,~
10943   code-for-last-row,~
10944   columns-type,~
10945   columns-width,~
10946   corners,~
10947   create-extra-nodes,~
10948   create-medium-nodes,~
10949   create-large-nodes,~
10950   extra-left-margin,~
10951   extra-right-margin,~
10952   first-col,~
10953   first-row,~
10954   hlines,~
10955   hvlines,~
10956   hvlines-except-borders,~
10957   l,~
10958   last-col,~
10959   last-row,~
10960   left-margin,~
10961   light-syntax,~
10962   light-syntax-expanded,~
10963   name,~

```

```

10964     no-cell-nodes,~
10965     nullify-dots,~
10966     pgf-node-code,~
10967     r,~
10968     renew-dots,~
10969     respect-arraystretch,~
10970     right-margin,~
10971     rounded-corners,~
10972     rules~(with~the~subkeys~'color'~and~'width'),~
10973     small,~
10974     t,~
10975     vlines,~
10976     xdots/color,~
10977     xdots/shorten-start(+),~
10978     xdots/shorten-end(+),~
10979     xdots/shorten(+)-and~
10980     xdots/line-style.
10981 }
10982 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10983 {
10984     Unknown~key.\\
10985     The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
10986     \{NiceTabular\}. \\
10987     That~key~will~be~ignored. \\
10988     \c_@@_available_keys_str
10989 }
10990 {
10991     The~available~keys~are~(in~alphabetic~order):~
10992     &-in-blocks,~
10993     ampersand-in-blocks,~
10994     b,~
10995     baseline,~
10996     c,~
10997     caption,~
10998     cell-space-bottom-limit,~
10999     cell-space-limits,~
11000     cell-space-top-limit,~
11001     code-after,~
11002     code-for-first-col,~
11003     code-for-first-row,~
11004     code-for-last-col,~
11005     code-for-last-row,~
11006     columns-width,~
11007     corners,~
11008     custom-line,~
11009     create-extra-nodes,~
11010     create-medium-nodes,~
11011     create-large-nodes,~
11012     extra-left-margin,~
11013     extra-right-margin,~
11014     first-col,~
11015     first-row,~
11016     hlines,~
11017     hvlines,~
11018     hvlines-except-borders,~
11019     label,~
11020     last-col,~
11021     last-row,~
11022     left-margin,~
11023     light-syntax,~
11024     light-syntax-expanded,~
11025     name,~
11026     no-cell-nodes,~

```

```

11027 notes~(several~subkeys),~
11028 nullify-dots,~
11029 pgf-node-code,~
11030 renew-dots,~
11031 respect-arraystretch,~
11032 right-margin,~
11033 rounded-corners,~
11034 rules~(with~the~subkeys~'color'~and~'width'),~
11035 short-caption,~
11036 t,~
11037 tabularnote,~
11038 vl原因,~
11039 xdots/color,~
11040 xdots/shorten-start(+),~
11041 xdots/shorten-end(+),~
11042 xdots/shorten(+)-and~
11043 xdots/line-style.
11044 }
11045 \@@_msg_new:nnn { Duplicate~name }
11046 {
11047 Duplicate~name.\\
11048 The~name~' \l_keys_value_tl 'is~already~used~and~you~shouldn't~use~
11049 the~same~environment~name~twice.~You~can~go~on,~but,~
11050 maybe,~you~will~have~incorrect~results~especially~
11051 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
11052 message~again,~use~the~key~'allow-duplicate-names'~in~
11053 ' \token_to_str:N \NiceMatrixOptions '.\\
11054 \bool_if:NF \g_@@_messages_for_Overleaf_bool
11055 { For~a~list~of~the~names~already~used,~type~H~<return>. }
11056 }
11057 {
11058 The~names~already~defined~in~this~document~are:~
11059 \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
11060 }
11061 \@@_msg_new:nn { caption-above~in~env }
11062 {
11063 The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
11064 That~key~will~be~ignored.
11065 }
11066 \@@_msg_new:nn { show-cell-names }
11067 {
11068 There~is~no~key~'show-cell-names'~in~nicematrix.\\
11069 You~should~use~the~command~\token_to_str:N \ShowCellNames\
11070 in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
11071 \CodeAfter. \\
11072 That~key~will~be~ignored.
11073 }
11074 \@@_msg_new:nn { Option~auto~for~columns-width }
11075 {
11076 Erroneous~use.\\
11077 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
11078 That~key~will~be~ignored.
11079 }
11080 \@@_msg_new:nn { NiceTabularX~without~X }
11081 {
11082 NiceTabularX~without~X.\\
11083 You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
11084 However,~you~can~go~on.
11085 }
11086 \@@_msg_new:nn { Preamble~forgotten }
11087 {

```

```

11088     Preamble~forgotten.\\
11089     You~have~probably~forgotten~the~preamble~of~your~
11090     \@@_full_name_env: . \\
11091     This~error~is~fatal.
11092 }
11093 \@@_msg_new:nn { Invalid~col~number }
11094 {
11095     Invalid~column~number.\\
11096     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11097     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
11098 }
11099 \@@_msg_new:nn { Invalid~row~number }
11100 {
11101     Invalid~row~number.\\
11102     A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
11103     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
11104 }
11105 \@@_define_com:NNN p ( )
11106 \@@_define_com:NNN b [ ]
11107 \@@_define_com:NNN v | |
11108 \@@_define_com:NNN V \| \|
11109 \@@_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>	52
10	The environment <code>{NiceArrayWithDelims}</code>	56
11	Construction of the preamble of the array	61
12	The redefinition of <code>\multicolumn</code>	78
13	The environment <code>{NiceMatrix}</code> and its variants	95
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	96
15	After the construction of the array	97
16	We draw the dotted lines	104
17	The actual instructions for drawing the dotted lines with <code>TikZ</code>	119
18	User commands available in the new environments	125
19	The command <code>\line</code> accessible in code-after	131
20	The command <code>\RowStyle</code>	133
21	Colors of cells, rows and columns	136
22	The vertical and horizontal rules	148
23	The empty corners	165
24	The environment <code>{NiceMatrixBlock}</code>	167
25	The extra nodes	169
26	The blocks	173
27	How to draw the dotted lines transparently	200
28	Automatic arrays	200
29	The redefinition of the command <code>\dotfill</code>	202
30	The command <code>\diagbox</code>	202

31	The keyword <code>\CodeAfter</code>	203
32	The delimiters in the preamble	204
33	The command <code>\SubMatrix</code>	205
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	214
35	The commands <code>HBrace</code> et <code>VBrace</code>	217
36	The command <code>TikzEveryCell</code>	220
37	The command <code>\ShowCellNames</code>	222
38	We process the options at package loading	223
39	About the package underscore	225
40	Error messages of the package	225